

IEEE

# MICRO

DECEMBER 1990

Chips, Systems, Software, and Applications

**ESPRIT:**

Parallel systems  
mesh  
with industry needs



## F E A T U R E S

- |   |  |
|---|--|
| <p><b>8 Guest Editors' Introduction: Parallel Computing in Europe</b><br/><i>Jean-Francois Omnes, Thierry Van der Pyl, and Philip Treleaven</i></p> <hr/> <p><b>12 Parallel Computers for Advanced Information Processing</b><br/><i>Pierre H.M. America, Ben J.A. Hulshof, Eddy A.M. Odijk, Frans Sijstermans, Rob A.H. van Twist, and Rogier H.H. Wester</i><br/>Six major companies combined talents with several research organizations to investigate and compare design approaches for parallel computer systems.</p> <hr/> <p><b>16 Transputers—Past, Present, and Future</b><br/><i>Colin Whitby-Stevens</i><br/>Enhancing performance of existing transputers, the ESPRIT projects continue to develop the all-around capabilities of this chip—with a vision for its future in general-purpose computing.</p> | <p><b>20 The European Declarative System, Database, and Languages</b><br/><i>Guy Haworth, Steve Leunig, Carsten Hammer, and Mike Reeve</i><br/>To address future demands of immense, complex databases, this intelligent information server exploits large-scale parallelism and supports current interfaces such as Unix and SQL.</p> <hr/> <p><b>24 Architectures Within the ESPRIT SPAN Project</b><br/><i>Peter Rounce and Jose Delgado</i><br/>To integrate symbolic and numeric computing on parallel systems, project participants developed a target architecture that resulted in a number of significant advancements in programming languages and architecture.</p> <hr/> <p><b>28 Pygmalion: ESPRIT II Project 2059, Neurocomputing</b><br/><i>Bernard Angeniol</i><br/>Neuron processors, dedicated ASICs, and standard computational tools for programming and simulation could lead to a general-purpose parallel neurocomputer.</p> <hr/> <p><b>32 Annual Index, Volume 10</b></p> |
|---|--|

Cover by TIB/West © Michael Melford,  
Design & Direction

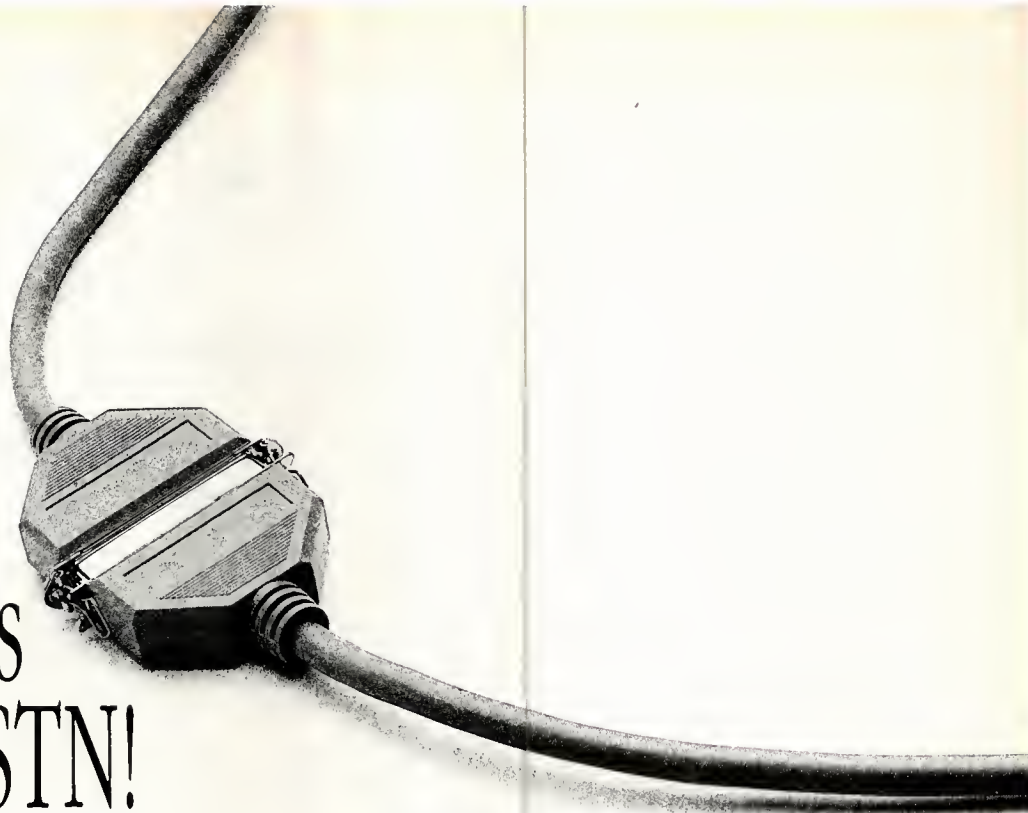
## D E P A R T M E N T S

- |  |  |
|--|--|
| <p><b>3 From the Editor-in-Chief</b></p> <p><b>5 Micro World</b><br/>Reunifying East and West</p> <p><b>11 1991 Editorial Calendar</b></p> <p><b>39 Micro Law</b><br/>The <i>Paperback</i> case, Part 2</p> <p><b>42 Micro Standards</b><br/>A standard to consider</p> <p><b>46 Micro Review</b><br/>Desk planners plus</p> | <p><b>48 Software Report</b><br/>Quality improvement</p> <p><b>51 Micro News</b><br/>Analyzing 32-bit computers; MCMs</p> <p><b>53 New Products</b><br/>DSP, design tools, hand-held computers</p> <p><b>59 Product Summary</b></p> <p><b>104 Micro View</b><br/><i>AMD v. Intel</i></p> |
|--|--|

Call for papers, p. 4; Reader Interest/Service/Subscription cards, p. 64A; Advertiser/Product Index, Change-of-Address form, p. 60; Computer Society information, Cover 3



# Make Great Connections Online On STN!



On STN International® you can access databases covering every area of engineering. You'll find bibliographic and numeric files produced by leading scientific organizations, like AIChE, Engineering Information, IEEE, National Institute of Standards and Technology, and many others.

And everything about STN has been created to assist you in obtaining this information efficiently and economically. On STN, you'll find special features which enhance your searching, whether you're a novice or an expert.

## **One Command Language —**

Using a few simple commands, you'll be able to obtain information in more

than 100 databases on STN. And STN's software, Messenger®, enables you to carry a search created in one database over to another on STN for further information.

## **Element Term Index —**

Through STN's Element Term (ET) Field, you can search for chemical symbols and other specialized notations. Using the ET Field can increase your accuracy AND efficiency.

## **Numeric Searching —**

On STN, you can search numeric values to locate substances having the property values you specify; search numeric ranges to find data you might otherwise miss; choose from SI, metric, engineering, or other units to

display property values; search with substance names, names of properties, or CAS Registry Numbers® to retrieve numeric data.

As an STN Customer, you'll get help through workshops, tutorial diskettes, a toll-free Help Desk, and newsletters. There's even a special Search Service available, for when you're short on time or have a backlog of work. No one supports you like STN.

Make great connections on STN by filling out and returning the coupon below.

Reader Service Number 1

☐ **YES! Please tell me how to become a user of STN International.**

Name \_\_\_\_\_

Title \_\_\_\_\_

Organization \_\_\_\_\_

Address \_\_\_\_\_

City, State ZIP \_\_\_\_\_

Phone \_\_\_\_\_

MAIL TO: STN International, c/a Chemical Abstracts Service, Marketing Dept. 30091,  
P.O. Box 3012, Columbus, OH 43210  
FAX TO: 1-614-447-3713





**IEEE Computer Society**  
PO Box 3014  
Los Alamitos, CA 90720-1264  
(714) 821-8380

**Circulation:** *IEEE Micro* (ISSN 0272-1732) is published bi-monthly by the IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; IEEE Computer Society Headquarters, 1730 Massachusetts Ave., NW, Washington, DC 20036-1903; IEEE Headquarters, 345 East 47th St., New York, NY 10017. Annual subscription: \$19 in addition to IEEE Computer Society or any other IEEE society member dues; \$35 for members of other technical organizations. Back issues: \$10 for members; \$20 for non-members. This journal is also available in microfiche form.

**Postmaster:** Send address changes and undelivered copies to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Second-class postage is paid at New York, NY, and at additional mailing offices.

**Copyright and reprint permissions:** Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limits of US Copyright Law for private use of patrons those post-1977 articles that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through the Copyright Clearance Center, 29 Congress St., Salem, MA 01970. Instructors are permitted to photocopy isolated articles for noncommercial classroom use without fee. For other copying, reprint, or republication permission, write to Permissions Editor, *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. Copyright © 1990 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved.

**Editorial:** Unless otherwise stated, bylined articles and descriptions of products and services reflect the author's or firm's opinion; inclusion in this publication does not necessarily constitute endorsement by the IEEE or the IEEE Computer Society. Send editorial correspondence to *IEEE Micro*, PO Box 3014, Los Alamitos, CA 90720-1264. All submissions are subject to editing for style, clarity, and space considerations.

---

#### EDITOR-IN-CHIEF

Joe Hootman  
*University of North Dakota\**

---

#### EDITORIAL BOARD

John Crawford  
*Intel Corporation*

David K. Kahaner  
*National Bureau of Standards*

Ashish Khan  
*Mips Computer Systems*

Hubert D. Kirmann  
*Asea Brown Boveri Research Center*

Richard Mateosian

Ken Sakamura  
*University of Tokyo*

John L. Schmalzel  
*University of Texas at San Antonio*

Michael Slater  
*Microprocessor Report*

John W. Steadman  
*University of Wyoming*

Richard H. Stern

James H. Tracey  
*University of Texas at San Antonio*

Philip Treleaven  
*University College London*

Carl Warren  
*McDonnell Douglas Space Systems*

Maurice Yunik  
*University of Manitoba*

---

#### MAGAZINE ADVISORY COMMITTEE

Jon T. Butler (chair)	Sushil Jajodia
B. Chandrasekaran	Ted Lewis
Manuel D'Abreu	H.T. Seaborn
James J. Farrell III	Bruce D. Shriver
Joe Hootman	Pradip K. Srimani
Anil Jain	John Staudhammer

---

\* Submit six copies of all articles and special-issue proposals to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202; (701) 777-4331; Compmail: j.hootman

---

#### PUBLICATIONS BOARD

Sallie Sheppard (chair)  
James J. Farrell III (vice chair)  
Dharma P. Agrawal  
Victor Basili  
P. Bruce Berra  
J. Richard Burke  
Jon T. Butler  
J. T. Cain  
B. Chandrasekaran  
David Choy  
James Cross  
Manuel D'Abreu  
Joe Hootman  
Sushil Jajodia  
Glen G. Langdon  
Ted Lewis  
Ming T. Liu  
C.V. Ramamoorthy  
H.T. Seaborn  
Bruce D. Shriver  
John Staudhammer  
Harold Stone  
Steven L. Tanimoto  
Joseph Urban  
Ben Wah  
Ron Williams

---

#### STAFF

H.T. Seaborn  
*Publisher*

Marilyn Potes  
*Editorial Director*

Douglas Combs  
*Assistant Publisher*

Marie English  
*Managing Editor*

Christine Miller  
*Staff Editor*

Don Toshach  
*Assistant Editor*

Pat Paulsen  
*Assistant to the Publisher*

Jay Simpson  
*Art Director*

Joseph Daigle  
*Design/Production*

Christina Champion  
*Membership/Circulation Manager*

Heidi Rex, Marian Tibayan  
*Advertising Coordinators*



## From the Editor-in-Chief



### Hellos and farewells

**T**his issue contains *IEEE Micro's* yearly account of the developments in the European computer industry. Because of its multinational environment, the European community approaches research and development in a unique way. It uses the ESPRIT program as a way of developing a technology base in the area of computers. An excellent example of co-operation, this program could well be emulated by other industries and countries interested in entering the high-technology marketplace.

A majority of the articles in this issue discuss accomplishments of the ESPRIT program. I think you will agree that guest editors Jean-Francois Omnes, Thierry Van der Pyl, and Philip Treleaven have done an excellent job of soliciting the articles.

Also, with this issue we welcome a new Editorial Board member and say good-bye to three others: Marlin Mickle, Yoichi Yano, and Steve Dyer. Marlin Mickle has long been the editor of the New Products department, and Yoichi Yano has helped in getting articles for the annual Far East issues. A special thanks go to Steve Dyer, who has been involved in the several special issues on digital signal processing and who has been an active board member for the past four years.

We welcome Ashis Khan to the Editorial Board. Ashis works with Mips Computer Systems, and he has been active in the organization of several conferences. Ashis will be a valuable addition to *Micro's* Editorial Board. His biography appears in *Micro News* this issue.

I have been associated with *IEEE Micro* for the past six years, either as an Associate Editor-in-Chief or as the EIC. During this time *Micro* matured into a magazine of some considerable standing. As with most jobs, however, the time

has come for me to step aside and turn over the reigns to the next caretaker.

My job was made easy by having an extremely competent, efficient managing editor, Marie English, and staff editor, Christine Miller. The Editorial Board members continually contributed a great deal to the quality of the magazine; they edited special issues and handled the reviewing of the vast majority of the articles that we published. One of *Micro's* strong points has been its multitude of departments and the quality of the writing that goes into each one. All of the department editors are quality people who have done an outstanding job. The success of this publication can be traced to all of these individuals. I was fortunate to be able to work with them.

*Micro* supporters have always been willing to try new and innovative ideas, and the selection of a someone located outside the US to serve as the new Editor-in-Chief continues this tradition. As far as I know, neither the Computer Society nor the IEEE has had an EIC who is not based in the States. Dante Del Corso has served as co-guest editor of two issues and lists among his accomplishments extensive academic publications. He is a capable engineer and will do an outstanding job as *Micro's* Editor-in-Chief. I look forward to the future issues, and I cannot think of a better person to be *Micro's* EIC than Dante Del Corso. I believe you will come to feel the same.

### In the mailbag

#### February 1990

"I liked [the] special [issue] on Hot Chips. In general, it is encouraging to see the trend to practice more than [to] academic [articles]. Educational papers have been excellent—keep them up." L.M., Caracas, Venezuela (The Hot Chips issues of *Micro* were two of the most popular issues we have published in some time.—J.H.)

"I liked [the] article on the i486 CPU. Excellent!!" D.F., Argentina

"I liked this issue—an excellent one. 'The Daily Micro' on the cover: Is it fictitious or a reality? I've not seen one." V.R.M., Bangalore, India (As far as I know, "The Daily Micro" shown on this cover is not published.—J.H.)

#### April 1990

"I liked [the] announcement of the gallium arsenide MC68030 board by

General Micro Systems on p. 91. (1st April or dream come true?) G.M., Warsaw, Poland (This announcement was received from Motorola and was not an April Fool's joke!—J.H.)

"I liked Micro View by Michael Slater. Is he trying to say that hardware is behind software? I would like to see material on commercial products." F.F., Tehran, Iran

"I would like to see information about the IBM RISC system." A.A.Y., Elhadara, Egypt

#### June 1990

"I would like to see [an] occasional elementary explanation, at [the] kindergarten level, of commonly used techniques—for those of us who have drifted into specialized tracks." J.E.T., Monce, IL (One of the goals that I wanted to accomplish as *Micro's* EIC

was to publish more tutorial material in the magazine. To date, I have not been able to accomplish this. Writing a tutorial is a difficult task, and so is reviewing them.—J.H.)

"I would like to see more on microcomputer networking, please!" S.K., Santa Clara, CA

"I think you publish a good mix of papers; I do enjoy reading papers written by people who have designed the products they are discussing. They have an opportunity to say why they did what they did in a way that does not always appear in instruction books, application notes, etc. I have not had difficulty identifying authors who are working for the company mentioned. S.H., Buffalo, NY

# Call for Papers

## IEEE Micro seeks manuscripts for 1991 and 1992

Submit manuscripts to:  
Joe Hootman, Editor-in-Chief, EE Dept.,  
University of North Dakota,  
PO Box 7165, Grand Forks, ND 58202,  
phone (701) 777-4331.

### GENERAL-INTEREST TOPICS

- Biological computing
- Artificial intelligence
- VHDL design and workstations
- Operating systems
- Multiprocessing
- Microcomputing to aid the handicapped
- Optical computing





**Hubert Kirrmann**

*Asea Brown Boveri*

*Research Center*

*CRBC.1*

*CH-5405 Baden,*

*Switzerland*

## Reunifying East-West electronics industries

**I**t is hard for the best pupil in one class to transfer to a more advanced class. Often this once-successful student makes sub-average grades in the new class. The struggle to catch up takes hard work, evening study, and a lot of courage.

Today, this is the experience of East German engineers. A year ago, they set the standards for the Eastern European electronics industry. Their engineers helped to make space exploration possible. Their electronics industry—represented by the state-owned VEB (Volks-Eigene Betriebe, or people-owned factories)—maintained a monopoly on electronic appliances and telecommunications. The VEB Mikroelektronik Kombinat manufactured 4-Mbit DRAMs (dynamic RAMs) and 32-bit processors. ASICs (application-specific integrated circuits) came in 1.5- $\mu$ m CMOS (complementary metal-oxide semiconductors). The VEB Robotron Kombinat produced about 100,000 personal computers per year.

Generous support of the former German Democratic Republic (GDR) government made the electronics sector the flagship of its entire national industry. In fact, the Communist government aimed to make the electronics industry autonomous within a few years.

Probably no other country in the world invested so heavily per inhabitant in microelectronics. With more than 150,000 employees in both the microelectronics field and the electronics industry, East Germany rivaled West Germany in numbers of people working in electronics-related areas.

Upon further investigation, however, one finds these statistics a little misleading. For example, 64-Kbit DRAMs and 8-bit, Z-80 lookalikes make up the bulk of East German series production. Half of the IBM compatibles produced are still

8-bit machines. The Robotron computers copy the IBM 370 and VAX 11/780, which lag 10 years behind the times. Compared to West Germany, East German component technology lags an estimated five years behind West Germany, which itself is not a world leader. Appliances and process control equipment are a decade behind; in some cases the technological gap with Western Europe spans nearly 15 years.

As the Berlin Wall tumbled down, one immediately perceived the GDR's inability to compete with the electronics industry in the West. In spite of massive price slashes, East German citizens refused to buy otherwise acceptable radios built in their country because of the radios' appearance, quality record, and the reminiscence of bad times.

### An example

The competition and reunification process of the two German industries is beautifully illustrated by the Osram light bulb factory in Berlin. Split in two by the building of the two Germanies in 1949, each of Osram's half-firms worked independently for 40 years, supplying goods to each of their respective markets. In December 1989, prior to the reunification of the countries, Osram executives evaluated reuniting the firms.

Employees of one firm visited the other. The first question of the Eastern visitors was, "Where are the people?" Osram's fully automated production facility in West Berlin needed no production personnel except for maintenance and supervisory control workers. In contrast, the East Berlin plant resembled a crowded marketplace. These employees accomplished production by hand, even using machines of 1949 vintage. Each Eastern factory worker achieved only one tenth of the production of the Western factory employees.

Poor quality and lack of quantity resulted. The West Berlin employees could not believe that people worked under such conditions. They also wondered about employee salary levels that allowed the East Berlin firm to sell the bulbs so cheaply.

Under these conditions, the West Osram declared—contrary to its initial statements—that reuniting with the Eastern firm entailed employee layoffs and reduction of the considerable benefits to workers (including free medical care and free kindergarten). Around the time of this announcement, the introduction of the free exchange of goods brought an influx of Western goods into East Germany. Sales of East Osram bulbs plummeted to zero—even with dramatically slashed prices—because East Germans preferred West German bulbs.

Consequently, the Eastern factory ceased operations. West Osram refused to invest in it or absorb it. Covering the Eastern market required only a moderate increase in the firm's automated facilities, and West Osram did not desire the know-how of East German engineers.

The world crumbled for all East Osram employees: no more job security, loss of social support, and soaring cost-of-living rates to West German levels. Many former employees emigrated to the West—the younger workers training for new vocations, the older ones hoping the West German social security net would adequately support them.

The East and West Osram situation illustrates the massive shake-up occurring with German reunification. On the other hand, Westerners now wonder how much they will pay for this process.

The press analyzes the extreme divergence of the two Germanies either by saying, "I always said it would end up this way" or "They would have made it if...." East Germans themselves provide the best analysis. Professor Wolfgang Marschall of the Academy of

Science of the GDR presents an excellent case for his position.<sup>1</sup> The East German analysts agree that the lack of pressure from their country's market allowed uneconomical firms to develop. Low per-capita productivity resulted because full employment received a higher priority than productivity. Government subsidies for the flagship electronics industry occurred at a cost to other national industries. Planning ignored the rules of international trade. Isolation from the Western market and the scientific community also played important roles in the breakdown.

### **East meets West**

In the electronics industry, East German engineers unquestionably match the skill of their Western colleagues. In fact, considering the poor conditions under which the Easterners work for years, one can only be amazed that the technology they developed lags behind the West by only a decade and a half. The quality of teaching and the theoretical work at the Academy of Science compares favorably to Western standards—even without rich endowments.

Contrary to some opinions, the motivation of East German engineers remained high even before talk of reunification. Instead of satisfying economic objectives, they optimized their work in a spirit of competition with the government plan. For instance, the designers cared little how much their products cost: figures for costs-per-transistor functions totaled 10 times higher than the international level, production of DRAMs yielded below 10 percent, and production of 32-bit microprocessors yielded under 0.01 percent.

VEB maintained uneconomical production lines simply because the government plan imposed them to supply a few special parts, such as those used by the army. Although perfectly conscious of the economic nonsense, the executives saw no other alternative

since their plant almost solely supplied the Eastern market.

Indeed, although the international market stocked most of these specialized parts, NATO's Cocom (coordination committee) embargo closed the free market to East Germany. Ironically, Cocom not only restricted the East German electronics industry from purchasing parts from the West, it allowed the industry to subsist without an economical basis.

Of course, when firms really needed Western computers and components, they received them through a rather tedious process. We know now that the GDR's special department, the KoKo (Kommerzielle Koordination), used the "gray market" to supply East German firms with items prohibited by Cocom. Use of this channel resulted in expensive supplies and slower development of technology. For instance, one can trace the poor yields of DRAMs to a successful Cocom action to prevent the GDR from obtaining semiconductor manufacturing machines from the West.

The East German development departments lacked modern design tools and advanced components. Advanced components, if produced at all, ended up going to the military. Lack of currency and export restrictions hampered the acquisition of tools and components from the West. English language barriers kept the GDR engineers from obtaining fresh news about their field (Russian is the usual second language). They relied on West German electronic magazines as their prime source of information. The general fear of speaking openly about one's work hindered the interchange of information—especially with Russian scientists.

Even now, the VEB engineers ignore how many laboratories in Russia work in the same domain as themselves. Also, the regime reserved participation in Western conferences for its loyalists, whose preoccupation with smuggling goods took precedence over exchanging information.



The East German labs generated many good ideas, but translating ideas into a market product was an exception. By comparison, Marschall estimates Japan's transfer efficiency is "70 times better" than East Germany's. In fact, GDR politicians viewed translation to products as unimportant; they preferred to boast about their high-technology, 4-Mbit RAMs than about market sales.

Actually, sales were good, but customers in the Eastern European (Council for Mutual Economic Assistance, or Comecon) countries paid in soft currencies, while technology purchases required hard currencies. The "sales" departments ended up as "customer keep-away" departments, since demand far exceeded production. As a result, the firm determined the priority of its customers.

---

***East Germany's  
estimated  
demand for  
personal  
computers totals  
150,000-200,000  
machines a year.***

---

The lack of building maintenance, the struggle to acquire parts, poor telephone service, and the unreliable post office comprised a large portion of daily work. Administrative red tape and the dust of bureaucracy added to this gloomy picture. One learns that a country cannot sustain a high-tech industry amidst a broken-down economy. High-tech industries depend on a functioning infrastructure. Many developing countries make the same mistakes.

The responsibility for this situation falls not entirely on the former GDR

government, but rather reflects the failure of a system. Before speaking of the superiority of one system over another, one should remember that the symptoms and problems experienced by the East Germans also arise in Western industries, especially in large conglomerates. In these organizations, the responsibility of individuals often blurs and strategic decisions sometimes defy common sense. Every now and then, the invoking of arguments similar to those of the Eastern government hinders the free interchange of information and goods. The difference is this: Market laws in the West rapidly eliminate unhealthy firms. Finally, one must recognize that the East German industry collapsed not just because of its own evils. The technology blockade and the diversion of resources for the arms race at least worsened the situation significantly.

#### **What will happen next?**

The rejuvenation of Eastern firms seems quite uncertain. After a first phase of enthusiastic talks of cooperation, Western companies now either ignore Eastern firms or buy them to secure market share. As a result, few Eastern firms will survive. The microelectronics and computer industries may stay in business because, apart from the optical industry, they are the only sectors that attain international standards. However, the future of VEB Mikroelektronik remains clouded, even if it gains membership in the European Semiconductor Jessi (Joint European Submicron Silicon) project. One expects massive layoffs at VEB.

The Eastern engineers will help to fill the current shortage of design engineers in West Germany. Similar to what Czech engineers experienced after their defection to the West during the 1968 uprising in Prague, qualified East German engineers will obtain key positions in the Western industry within a few years. They bring a resourcefulness not often found in the West.

The market of East Germany will

expand greatly in the next several years. For instance, estimated demand for personal computers totals 150,000 to 200,000 machines a year. It will take cooperation between Western companies and Eastern firms (now likely operating as sales offices) to satisfy this demand.

Interest in the German reunification goes beyond the two affected countries. Reunification presents a laboratory for people to study the possible reunification of Europe, including most of the Soviet Union—a market second only to China in terms of population. Europe faces its second post-World War II reconstruction, and East Germany may become the bridge between East and West.

---

#### **Reference**

1. W. Marschall, "DDR-Mikroelektronik—Mittlerrolle zwischen Ost und West?" ("Microelectronics in the German Democratic Republic: A Negotiator Role Between East and West?"), *Elektronik*, No. 14, July 6, 1990, pp. 36-51.

---

#### **Reader Interest Survey**

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 174    Medium 175    High 176

---



TIB/West/Michael Melford © 1990

## Guest Editors' Introduction

# Parallel Computing in Europe

Jean-Francois Omnes

Thierry Van der Pyl

**CEC ESPRIT**

Philip Treleaven

**University College  
London**

**T**he European Economic Community supports a large number of research projects investigating the design of parallel computing systems. The European Strategic Programme for Research and Development in Information Technology funds many of these projects.<sup>1,3</sup> ESPRIT has the following objectives:

- to provide the European information technology industry with the basic technologies needed to meet the competitive requirements of the 1990s,
- to promote European industrial cooperation in information technology, and
- to pave the way for standards.

Projects selected from public calls for proposal and based upon an annually updated Workplan implement ESPRIT's goals. The program comprises collaborative, precompetitive research and development projects, which are carried out across frontiers by Community companies, universities, and research institutes. The European Community budget funds 50 percent of a project, and project participants fund the other 50 percent.

A consortium of companies and universities, each containing at least two independent industrial partners from different member states, undertakes each project. Projects vary greatly in composition. A large project might involve three companies and three universities from different member countries of the European



Community, with funding of 10 million ECUs and a span of three years. (As of October 1990, 1 European Currency Unit equals 1.31 US dollars.) Projects subdivide according to focus into A-type projects that emphasize research and development, and B-type projects emphasizing longer term applied research.

The first phase of ESPRIT, called ESPRIT I,<sup>1,2</sup> was a five-year research program lasting from 1984 to 1989. The total research and development effort of the first phase amounted to approximately 1.5 billion ECUs in funding. ESPRIT I launched 226 projects. At its peak, 3,000 engineers and scientists from 420 independent organizations worked full-time on ESPRIT I projects.

The second phase, called ESPRIT II,<sup>2,3</sup> started in 1988 and continues with the same broad objectives. A larger scale operation, it still preserves the mechanisms used by ESPRIT I. These include cost-sharing between the European Community and partners, the principles of consensus building and on-going assessment, and the Workplan. The total ESPRIT II budget amounts to approximately 3.2 billion ECUs, or a project cost per year of approximately 800 million ECUs.

The first-call ESPRIT II proposals met with an unprecedented response. The total requested funding exceeded the available budget by a factor of eight. After an evaluation process by independent experts, ESPRIT II advisers selected 156 proposals, involving 585 participating organizations. A particularly encouraging sign comes from the interest of small and medium-size enterprises (SMEs) in ESPRIT. Between ESPRIT I and ESPRIT II the number of participating SMEs (defined as having less than 500 employees) rose from 180 to 290. Of these SMEs almost half employ less than 50 employees. Of the 315 independent information technology vendors participating in ESPRIT II, 65 percent qualify as SMEs.

In addition to its main research program, ESPRIT II maintains so-called Basic Research Actions, which concentrate on fundamental research. Basic Research Actions are small research projects mainly undertaken by universities and research institutes throughout Europe.

In Europe, ESPRIT is the focus for research into parallel computers and new programming languages. ESPRIT devotes about 10 percent of its resources to parallel systems. This issue of *IEEE Micro* presents five significant research projects on parallel computing. We chose these projects to reflect the diversity of approaches.

The first article by Odijk of the Philips Research Laboratories in The Netherlands describes ESPRIT project 415, entitled "Parallel Architectures and Languages for Advanced Information Processing—A VLSI-Directed Approach." This project,<sup>4,6</sup> which began in 1984 and ran for five years, investigated and compared the major approaches in designing high-performance parallel computer systems: object-oriented, functional, and logic. It brought together six major European information technology companies, Philips, AEG, GEC, Bull, CSELT,

and Nixdorf, which were supported by a number of outstanding research organizations.

Whitby-Stevens, in his article about the Inmos transputer, examines the continuing development of this family of microprocessors.<sup>7</sup> The transputer family fulfills four main objectives: It creates a product range that is easy to program and engineer; it provides maximum performance to the user; it utilizes increasing levels of VLSI integration; and it creates a programmable component for large parallel systems. This article begins by reviewing the basic parallel programming and architecture concepts of the transputer that led to the next-generation transputer, codenamed H1. It also outlines some of the ESPRIT projects that are assisting the transputer's development. One, the Supernode project, produced a range of parallel computers with up to 1,000 processors. The article also describes two innovative transputer-based applications; one is a hand-held satellite navigation system and the other, the Marconi Martello long-range radar system.

---

***In Europe, ESPRIT is the focus for  
research into parallel computers  
and new programming  
languages.***

---

The third article by Haworth, Leunig, Hammer, and Reeve, describes the European Declarative Systems project.<sup>8</sup> A major technology integration project within ESPRIT II, EDS brings together Bull, ICL, Siemens, their jointly owned ECRC research center, and many universities across Europe. A major goal of EDS is the production of a high-performance, parallel relational database server. The EDS system being developed will support business information across a spectrum from data to knowledge, and manage it intelligently. To this end, it will support Unix, extended SQL, Lisp, C++, and the Elipsys parallel logic programming language.

Rounce from University College London and Delgado of Inesc Portugal describe the SPAN project,<sup>9</sup> which investigated the integration of parallel symbolic and numeric processing. The project covered parallel applications, languages, and architectures. At the core was the so-called Kernel System, a target machine language (Parle), and its virtual machine. This core served as the focal point for compiling the symbolic and numeric languages onto parallel numeric and symbolic parallel machines. The article presents two architectures, Sprint, a parallel microprocessor, and DICE, the parallel object-oriented system, both developed as part of SPAN.

LASTLY, THE ARTICLE BY ANGENIOL, formerly of Thomson-CSF and now of Mimetics in France, presents the Pygmalion neural computing project.<sup>10</sup> The objectives of Pygmalion are 1) to demonstrate to European industry the potential of neural networks for applications in image, speech, and acoustic signal processing; and 2) to develop a European "standard" neural network programming system. This system comprises a graphical environment, library of common algorithms, and high-level language. Pygmalion brings together many of the leading neural computing research groups from European industry, research institutes, and universities. These include Thomson-CSF, CSELT, Philips, SEL Alcatel, Siemens, and universities from England, France, Portugal, and Spain. ■

Jean-Francois Omnes' biography and photograph were not available.

**Thierry Van der Pyl** is coordinator of the Advanced System Architectures subarea in ESPRIT. Since 1984 he has been on loan from the Centre National de la Recherche Scientifique (CNRS in France), where he worked as charge de recherche in the field of pattern recognition and artificial intelligence. He studied at the Ecole Normale Supérieure de l'Enseignement Technique and received his doctorate of state from the Université Pierre et Marie Curie, Paris.



**Philip Treleaven** is professor of computer science at University College London. His research interests include neural computing, parallel computer architecture, new forms of programming languages, and very large scale integration. He has worked with a number of ESPRIT projects, including the large ESPRIT project 415 and the ESPRIT Pygmalion neural computing project. He has also worked at the Fifth Generation Research Institute (ICOT) in Tokyo.

Treleaven holds a BTech degree from Brunel University and MSc and PhD degrees from Manchester University, England. He has published over 70 papers, lectured in over 25 countries, and been an invited speaker at a number of conferences including the International Conference on Fifth Generation Computer Systems in Tokyo. He is a member of the IEEE, the ACM, and the British Computer Society, for which he has served as a council member.

Direct questions concerning this article to Philip Treleaven, University College London, Gower Street, London WC 1E 6BT, United Kingdom.

## References

1. ESPRIT, "ESPRIT-87: Achievements and Impact," *Proc. Fourth Annual ESPRIT Conf.*, North-Holland, Amsterdam, 1987.
2. ESPRIT, "ESPRIT-88: Putting the Technology to Use," *Proc. Fifth Annual ESPRIT Conf.*, North-Holland, 1988.
3. ESPRIT, "ESPRIT-89," *Proc. Sixth Annual ESPRIT Conf.*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1989.
4. P.C. Treleaven, ed., "Parallel Computers: Object-Oriented, Functional, Logic," Wiley Series in Parallel Computing, John Wiley & Sons, New York, 1990.
5. J. de Bakker, ed., "Languages for Parallel Architectures: Design, Semantics, Implementation Models," Wiley Series in Parallel Computing, 1990.
6. W. Bronnenberg et al., "DOOM: A Decentralized Object-Oriented Machine," *IEEE Micro*, Vol. 7, No. 5, Oct. 1987, pp. 52-69.
7. M. Homewood et al., "The IMS T800 Transputer," *IEEE Micro*, Vol. 7, No. 5, Oct. 1987, pp. 10-26.
8. F. Andres et al., "EDS—Collaborating for a High Performance Parallel Relational Database," *Proc. ESPRIT-90*, North-Holland, Nov. 1990.
9. A.N. Refenes et al., "SPAN: Parallel Computer Systems for Integrating Numeric and Symbolic Processing," *Proc. ESPRIT-88: Putting the Technology to Use*, North-Holland 1988.
10. B. Angeniol and G. de La Croix Vaubois, "Pygmalion—Neurocomputing: ESPRIT II Project 2059," *Proc. Int'l Neural Networks Conf.*, Kluwer Academic Publishers, July 1990, pp. 1054-1057.

## Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 150

Medium 151

High 152



# IEEE MICRO

## 1991

# Editorial Calendar

### FEBRUARY

#### *Microprocessors in Education Special 10th Anniversary issue*

- Demonstrating new hardware to students
- Part 3, legal issues surrounding the *Lotus vs. Paperback* decision
- Industry prospects in the next 10 years

**Ad closing date: January 1, 1991**

### AUGUST

#### *General interest*

- HDTV
- Unix-based RISCs
- Communications
- Personal computers
- Optical computing in Japan

**Ad closing date: July 1, 1991**

### APRIL

#### *Special Far East issue*

- Microcomputer and RAM technology from Japan, Korea, Taiwan, and the Pacific Basin
- Software projects in Japan
- Current TRON Project offerings, Japan's effort to introduce a standard computer system

**Ad closing date: March 1, 1991**

### OCTOBER

#### *Computers in bioengineering*

- Medical imaging
- Robotics in surgery
- Monitoring and measuring biological processes
- Applications in this crucial computer field
- Special feature: Fuzzy logic projects in Japan

**Ad closing date: September 1, 1991**

### JUNE

#### *Hot Chips II Symposium*

- This extremely popular issue presents the latest developments in microprocessor and chip technology used to construct high-performance workstations and systems.
- Past contributors include: Intel, Motorola, Texas Instruments, Apple, Sun, Stardent, Weitek, Metaware

**Ad closing date: May 1, 1991**

### DECEMBER

#### *Database machine implementation*

- Ensuring that today's microcomputers efficiently implement databases
- Designing their architectures
- More to be added

**Ad closing date: November 1, 1991**

IEEE Micro helps designers and users of microprocessor and microcomputer systems explore the latest technologies to achieve business and research objectives.

Feature articles in IEEE Micro reflect original works relating to the design, performance, or application of microprocessors and microcomputers.

All manuscripts are subject to a peer-review process consistent with professional-level technical publications. IEEE Micro is a bimonthly publication of the IEEE Computer Society.

**Advertising information:** Contact Marian Tibayan, Advertising Department, IEEE Computer Society, PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380; fax (714) 821-4010.

Articles may change. Please contact editor to confirm.



# Parallel Computers for Advanced Information Processing

*ESPRIT project 415 involved the study of object-oriented, functional, and logic programming styles in six subprojects. We describe the parallel languages and architectures designed to implement them as well as our parallel object-oriented system. This design includes a novel language, decentralized memory architecture, and system software.*

Pierre H.M. America

Ben J.A. Hulshof

Eddy A.M. Odijk

Frans Sijstermans

Rob A.H. van Twist

Rogier H.H. Wester

**Philips Research  
Laboratories**

**P**arallel computing efforts are probably as old as the history of computing itself. For a long time performance requirements of numerical programs, for use in solving scientific and engineering problems, drove these efforts. In the early 1980s, however, research programs began to apply parallelism to the wide class of symbolic programs. The Fifth-Generation Computer System program in Japan aimed at a very ambitious leap toward an information society through the combination of artificial intelligence and parallel computing.

Europe soon followed and initiated its ESPRIT I program from 1984 to 1989. ESPRIT's distributed approach identified important topics and formed consortia to share with the scientific and industrial bodies in Europe. A significant part of this program concerned knowledge engineering and advanced architectures under the heading of advanced information processing (AIP). From the start, the European information technology industry actively participated.

In late 1984 the first and largest project on parallel symbolic computing started under ESPRIT number 415. The project carried the title of Parallel Architectures and Languages for Advanced Information Processing—a VLSI- (very large scale integration) oriented approach. Unlike most other projects, 415 did not dictate one parallel technology but aimed at investigating and comparing, in a language-first approach, the major approaches

in designing high-performance parallel systems for AIP: logic, functional, and object-oriented. Six of the 12 major European information technology companies, supported by a number of outstanding research organizations, carried out the project.

In each of six subprojects the researchers investigated a specific style of parallel processing. Their goals were to achieve a parallel implementation for a (novel) programming language and demonstrate the performance improvements in some selected symbolic applications. Prototypes were delivered, ranging from four to 100 processors and showing good speed-ups as well as absolute speeds.

Before we describe in detail the technology of the object-oriented approach pursued by our team, we offer an overview of project 415. We discuss its starting points, its approaches, and its results. In particular, you will find it interesting to see which common conclusions were reached among the different styles.

## Symbolic parallel computing

Numeric algorithms, characterized by regular data structures and uniform operations on them, benefit from the parallel execution means of vector computers. These algorithms also benefit from shared-memory multiprocessor systems, supported by parallelizing compilers and careful tuning by the programmer.

The new category of symbolic applications,



**Table 1. Partners, subcontractors, and approaches.**

Subproject	Partners	Subcontractors	Contributions
A	Philips	CWI, Amsterdam Technical University Eindhoven University of Oldenburg	Object-oriented machine Pooma
B	AEG GEC	Technical University of Berlin University College London Imperial College London	VLSI simulator on Pooma Lazy functional languages, Reduction machine
C	BULL	LITP, University of Paris ESIEE, Noisy-le-Grand	Logic database machine
D	CSELT	University of Pisa	Mixed-logic functional machine
E	Nixdorf	Stollmann, Hamburg	Dataflow machine
F	Nixdorf	LIFIA-IMAG, Grenoble Technical University of Munich	Connection method Logic machine

partly overlapping with the general denominator of artificial intelligence, requires programming styles and languages with improved expression power as well as more dynamic and flexible execution mechanisms. Additionally, high-level concepts of communication and synchronization that resulted from research in the 1970s needed to be integrated in such languages to achieve effective programming for and efficient exploitation of parallelism.

These new concepts and the desire for a scalability of at least two orders of magnitude of parallelism necessitated a broad scope of research. By scalability, we mean that the concepts hold for a large range of machine sizes (number of processors.) We researched the simultaneous and integral design of parallel architectures and languages for symbolic applications. Representative instances of the latter would prove the expressiveness of the languages and the achieved performance of the parallel systems. Finally, researchers recognized that a better understanding of the theoretical foundations of the parallel programming styles and concurrency in general would impact the quality of language and program design.

Project 415's charter, as reflected in its full title, acknowledges the above observations.

Project participants considered a number of programming models to be potentially feasible, but neither the validity nor even the superiority of one of them had been established. Therefore, they decided to investigate a number of distinct parallel programming styles and propose architectures and languages to support these styles. They defined subprojects, each to be executed by one partner and its academic subcontractors (see Table 1).

Projectwide working groups formed in the areas of architecture and applications, semantics, and proof theory and verification. The groups facilitated a proper platform for the presentation and discussion of mutually important topics and

advanced the theory of concepts involved. The project comprises some 280 man-years, 20 percent of which were dedicated to the working groups.

The project emphasized education and exchange of scientific results in this area, eventually causing two summer schools<sup>1,2</sup> and two Parle conferences<sup>3,4</sup> to be organized. The proceedings of the latter include reports on the subprojects.

Before describing the goals and technical directions of the subprojects, let's discuss the issues at stake in a general manner. In a language-first approach, the feasibility of the selected programming model forms the hypothesis for each of the approaches. An abstraction of the corresponding implementation is the execution model. Table 2 on the next page provides a survey of these and links a number of attributes.

Logic and functional programming models belong to the class of declarative systems, while object-oriented programming falls within the imperative programming class. Declarative languages are considered to be higher level, more expressive, and concise, and they allow formal assessment due to their strong mathematical basis. Until recently however, their efficient implementation has not been well understood.

Two categories of parallelism exist, explicit and implicit. In the latter case the implementation (compiler and operating system) extracts an amount of parallelism from the program. With explicit parallelism (for example, in object-oriented programming) the programmer holds responsibility for partitioning a program into objects. To avoid burdening the programmer, the language should offer high-level, natural facilities for communication and synchronization. Implicit parallelism is in principle present in the execution model of declarative languages. Thus, programmers would be able to concentrate on the algorithm per se. The compiler has to carry out the burden of extracting enough parallelism with a coarse-enough granularity.

Table 2. Execution models and attributes for subproject system.

Subproject	Language style, operational model	Parallelism	Architecture
A	Object-oriented, control flow	Explicit	Distributed memory, packet-switching network
B	Functional, graph reduction	Implicit	Emulation on transputer
C	Logic (many data), delta driven	Implicit	Distributed memory, bus based
D	Logic + functional, SLD resolution	Explicit	Shared memory, multistage network
E	Single assignment, dataflow	Implicit	Shared memory, bus based
F	Logic (many rules), connection method	Implicit	Shared memory, bus based

The preferred parallel architecture relates to the programming model and the required size and scalability of the parallelism. A general choice falls between shared-memory and distributed-memory architectures, and (mostly coinciding) between communication and synchronization through shared variables or message passing. Other choices relate to the amount of hardware support for specific features of the execution model. An important unifying concept in the project is that of the homogeneous parallel machine, which consists of a number of identical nodes. This concept provides for extensible systems and allows an implementation to take advantage of the relatively low replication costs of VLSI-based node realizations.

A common characteristic in the implementation of the various language models is the use of a dynamically varying number of processes. Usually, a processor carries out many processes, mostly with dynamically changing patterns of communication and interaction. This characteristic requires an operating system (kernel) to be resident on each of the nodes to manage the various resources at the node and system level.

Thus, a number of common and related problems arise, while their solution may differ with the language model. The subprojects directed their individual efforts at solving these problems in the contexts of the specific programming models.

### Goals and results of subprojects

Here, we describe each of the 415 subprojects on behalf of our partners, as indicated in Table 1.

**The object-oriented approach.** The essence of object-oriented programming is the subdivision of a system into objects, that is, integrated units of data and procedures. Objects communicate by passing messages, which must be interpreted as a request from the sending object to the receiving object to execute a certain procedure (called a method). To combine object-oriented languages with parallelism, we chose to associate with every object a process of its own. The model is intuitively appealing, with message passing as the only

facility for communication between objects.

Such a system's architecture would consist of many, functionally identical and control-flow structured, computers connected via a direct message-passing network. The computers contain a central processor, private memory, and a dedicated communication unit to perform the message passing without interference of the CPU. The concept allows systems of more than 1,000 such computers.

The major goals of this subproject (A in Table 2), derived from the chosen approach, follow:

- *A parallel object-oriented language POOL, in which significant application programs can be programmed.* The language provides the user with control of parallelism and granularity. The language must have clear, formal semantics. Support for the verification of programs is desirable, since it is even more important in a parallel environment than in a sequential one.
- *A prototype parallel object-oriented machine, Pooma, which consists of 100 identical self-contained computers.* Each computer has a powerful 32-bit processor, local memory, and communications means, which are connected in a direct packet-switching network. Since a Pooma node executes many (10 to 100) processes, the processor architecture must support multiprocessing. Each node of the system contains a copy of the operating system kernel. This kernel performs local resource management and cooperates with the other kernels for global operating system tasks. The prototype Pooma system connects as a satellite to a host computer, in which the programming environment resides. The prototype, based on existing technology, will offer facilities for experimenting and for evaluation of performance aspects.
- *Three significant applications in the area of symbolic processing that demonstrate the performance increase through parallelism on Pooma.* The first of these is a parallel theorem prover. The second is a parallel version



of the analytical component of the Rosetta natural language translation system. The Computer Science department at Philips Research currently develops Rosetta. The third application is a multilevel VLSI circuit simulator designed at AEG (described later).

We obtained these goals. Currently, a 100-node Pooma system is operational, based on the 68020 processor and a proprietary communication processor. The latter performs, in hardware, complete end-to-end routing for fixed-size packets, using alternative paths to avoid congestion. A built-in routing mechanism is free of deadlock. This powerful and efficient mechanism for higher layers of the system avoids interruption of the intermediate data processors.

POOL2 serves as the programming language for Pooma. A later version, POOL-X, a superset of POOL2, lets users program the database system of the Prisma project, also at Philips Research. In the course of the project, we defined both the operational and denotational semantics for POOL and proved that they are equivalent. Furthermore we designed a sound and complete formal proof system. The POOL implementation on Pooma consists of an operating system kernel and a compiler. Furthermore we developed a portable POOL implementation called *ptc*, which provides pseudoparallel execution on sequential systems and contains extensive support for debugging.<sup>5</sup> We implemented all compilers using *Elegant*,<sup>6</sup> a set of newly designed construction tools for parsers and code generators based on lazy evaluation in attribute grammars.

We obtained designs and implementations of a number of applications. Among these are the above-mentioned theorem prover and analytical component for natural language translation. First evaluations show promising speed-ups.

**A VLSI simulator in POOL.** Another main goal of subproject A is a new parallel, multilevel simulator (PMLS) for VLSI circuits.<sup>7</sup> PMLS runs on general-purpose parallel machines and combines multilevel simulation and exploitation of parallelism to achieve optimal performance. The initial implementation of the simulator is in the object-oriented POOL language for the parallel Pooma machine. Powerful simulators are a key element in today's and future digital circuit design environments and a significant application for future parallel machines.

PMLS focuses on the logic design levels—register transfer, functional gate, and switch—with provisions for including the programming and the electrical levels. The simulator uses one simulation concept for all levels. This broadband concept is characterized by distributed discrete-event simulation and the partitioning of the circuit into subcircuits, which subsimulator processes simulate in parallel. Each subcircuit may contain elements at different abstraction levels.

Subsimulators execute asynchronously using their own local time, communicate by exchanging event messages, and syn-

chronize by using the so-called time-warp approach. In this approach, a subsimulator always runs forward at full speed, but in case of an event message being received carrying a time stamp less than the local simulation time, the subsimulator rolls back to the earlier time. The concept delivers sufficient grain size to gain significant speed-up on a parallel general-purpose machine like Pooma. The object-oriented implementation allows the highly dynamic features of object creation and object interconnection for dynamic simulation—for example, incremental simulation, dynamic change of the abstraction level (zooming). It also provides high flexibility.

We obtained this goal. Currently PMLS is in the final implementation stage: We implemented the main parts of the simulator in POOL. Work continues on the advanced features (incremental simulation, zooming) and on the tuning of the simulator.

We use the POOL interpreter as well as the Pooma prototype to measure the execution behavior of the simulator. First evaluations show promising speed-ups. In addition, we've partially integrated the prototype version of PMLS into the existing environment of AEG's *Disim*, a commercial VLSI simulation system.

We therefore have a complete system, a parallel object-oriented program implemented on a parallel machine and partially integrated into the environment of a commercial simulation system. We've gone even beyond our original goal.

A further result of the project is a parallel fault simulator. In PFS, fault sets dynamically split into subsets, which then process as parallel jobs in the nodes of a local-area network, or LAN. PFS achieves a near-linear speed-up.

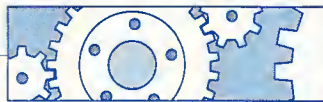
In summary we can say that VLSI simulation—as a special case of discrete-event simulation—is definitely a promising application of parallel general-purpose machines like Pooma.

**The functional approach.** In subproject B we chose to take a lazy functional language, which had no parallelism annotations, and implement it efficiently on a parallel architecture. At the beginning of the project, implementations of functional languages were interpretative, sequential, and slow. It seemed we could obtain significant increases in performance by 1) designing specialized hardware to support their execution, and 2) designing a parallel implementation that exploited their theoretically implicit parallelism.

We chose lazy functional languages because they seemed to be a very powerful programming paradigm, which is important for designing reliable and large systems. Furthermore, the parallelism was theoretically implicit in the execution mechanism of the language and thus invisible to the programmers.

We approached the problem in a "language-first" manner. We first developed a parallel evaluation model, which retained the semantics of the language and then drove the architectural design. The original project description speci-

*continued on p. 61*



# Transputers—Past, Present, and Future

---

*High-performance enhancements in transputers signal a trend toward general-purpose computing. We present the progress, products, and results of ongoing work in transputer development.*

---

Colin Whitby-Strevens

Inmos Limited

**A** transputer transformation is underway. Exploiting the multiprocessing potential of the transputer, ESPRIT projects continue to develop higher performance, lower cost parallel processing computers. In advancing the transputer toward general-purpose computing, Inmos Limited is also improving the transputer's suitability in embedded systems with the upcoming release of new products.

The first transputer emerged at a time when very large scale integration (VLSI) technology permitted a combination of a small, fast processor (20,000 transistors) and local memory and communications facilities on silicon. At the same time, the effective exploitation of VLSI technology also resulted in the development of reduced instruction-set computing (RISC) processors. Both the transputer and conventional RISC processors reevaluate architecture trade-offs in the context of VLSI capabilities. However, developers of the transputer created a design at the instruction-set level to support multiprocessing across a number of transputers and within a single transputer without the overheads usually associated with complex runtime software.

In embedded systems, the transputer appli-

cation designer directly controls the transputer hardware without the need for a resident operating system or runtime kernel. Applications for transputers include office systems (fax, videophones, laser printers, terminals), digital telecommunications, military systems and handheld satellite navigation systems (see box on sample applications), industrial control systems, and music synthesizers.

Initial promotion of the transputer focused on its use as a general-purpose component for special-purpose systems. A number of applications—particularly in graphics and image processing—clearly required high-performance, floating-point operations. Inmos achieved this capability very effectively by adding a floating-point unit within the transputer chip (in contrast to the conventional, but more cumbersome, coprocessor approach). This advancement opened up the exciting prospect of constructing parallel processing machines using arrays of transputers to produce supercomputer-level performance.

Applications requirements for simulation and modeling (for example, quantum chromodynamics and fluid-flow analysis) provided the incentive to establish the ESPRIT Supernode



## Two sample applications

The transputer's suitability to a wide range of embedded systems is illustrated by considering applications at both extremes. The first application—a hand-held navigational system—uses just one transputer to address such issues as high performance, low cost, and low power. The second application—a long-range, three-dimensional radar system—uses up to 4,000 transputers to provide supercomputer levels of performance, but in a real-time application.

### Hand-held satellite navigation system

Some 18 satellites, in six orbits, operate as part of a Global Positioning System. Each satellite continuously transmits its position in an encoded form, and by listening to four satellites one can calculate the position (in three dimensions) of the receiving station.

The traditional approach involves the use of complex dedicated signal processing hardware. However, the Gypsy hand-held receiver, developed by Columbus Positioning Limited, performs the complex signal processing and mathematics required in one transputer, which also services a keyboard and controls a display (see Figure A).

A large, developing market for the system includes marine navigation, transport control systems (reporting delivery truck positions), and automotive applications (dashboard navigation systems).

The advantages of the transputer-based solution over the traditional approach include relatively fast acquisition time (which also saves battery power), the elimination of custom logic for signal processing, the minimal amount of glue logic required, the low specification required on radio oscillators (transputer software performs frequency compensation), and quick implementation of future product changes.

### Long-range radar system

The Martello long-range, three-dimensional radar system under construction by Marconi Radar Systems uses up to 4,000 transputers for signal processing of the radar returns.<sup>1</sup> The transputers combine to form a very fast parallel computer that accepts digitized radar returns at its input and defines the range, bearing, and height of every target at its output. (Some environmental information is also added to the output.) The processing load amounts to roughly three billion operations per second through each radar channel.

The alternative to transputers—hardwiring—is inherently expensive (previous Martello systems used about 50 dif-



Figure A. Gypsy hand-held receiver.

ferent board types). With radar technology now moving so fast, hardwiring is doubly expensive because boards are quickly rendered out of date. By contrast, in the transputer system, software performs the entire signal processing task, and the same basic computer remains usable throughout a whole generation of radars.

The signal processing algorithms take place in real-time by using pipeline parallelism to spread the parts of the algorithms across an array, with computation overlapping communications (see Figure B on the next page). Simulation devised the optimum map, making it necessary to build only a small part of an array to prove the concept. In an array node of 50 transputers, housekeeping uses two transputers, while the rest are available for data processing.

*continued on p. 18*

## Two sample applications

continued from p. 17

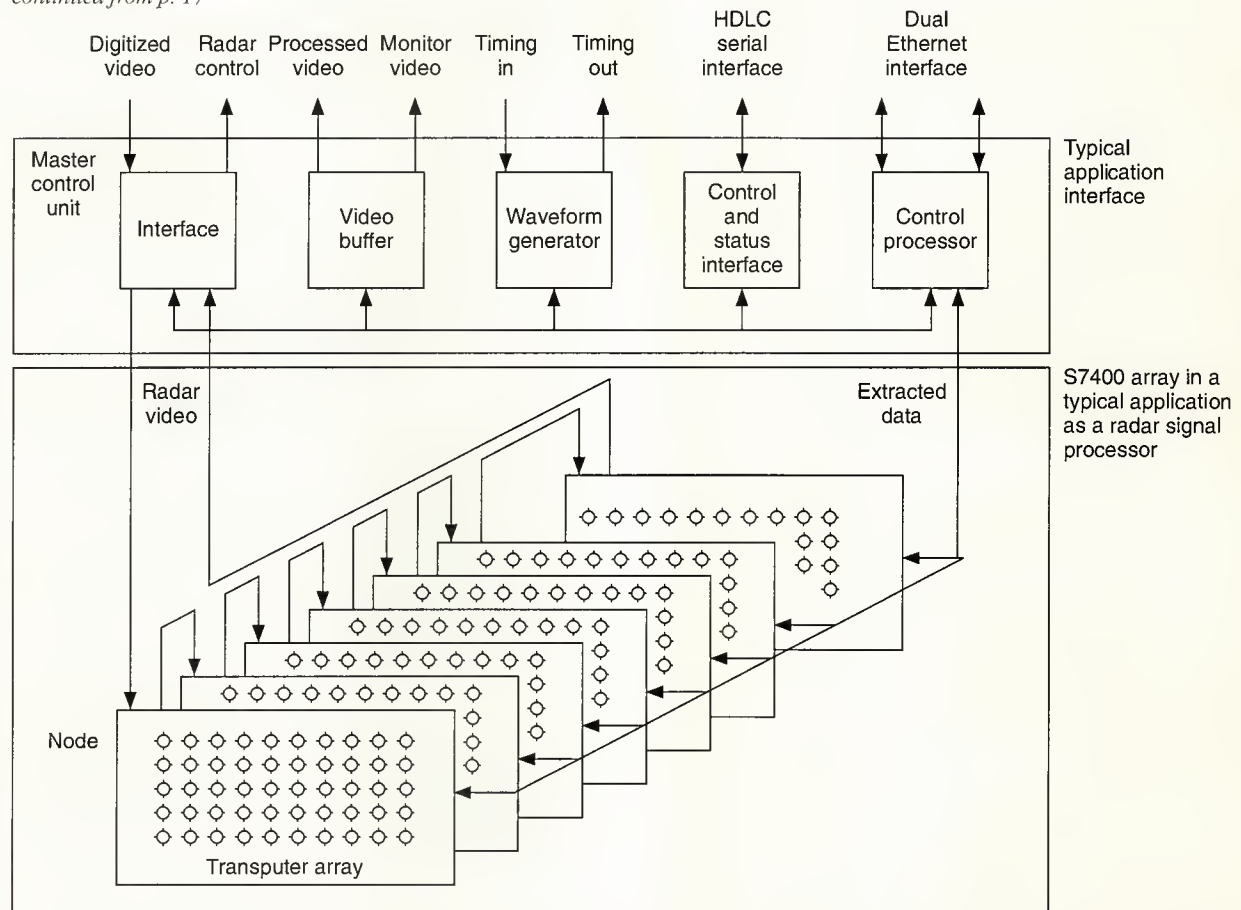


Figure B. Diagram of Martello radar system architecture.

Geometric parallelism achieves the processing rate required for the Martello radar. The radar coverage breaks into range bands, and a particular array node directs all samples from a given band. Each array node executes an identical program, but on different data. The scalable performance of the radar is proportional to the number of array nodes.

The vast number of processors in the Martello computer are 16-bit T222 transputers, with some 32-bit T425 transputers. A follow-on project proposes to construct a multifunctional radar for the Italian navy using floating-point transputers. However, no fundamental change to the architecture is necessary to take advantage of new generations of transputers.

project (see ESPRIT Supernode box). This project led to the development of many new commercial hardware and software products, including the Inmos T800 transputer, the Parsys Supernode machine, the Telmat T-node machine, and the N.A. Software Limited's parallel processing libraries for Fortran numerical routines. More significantly, the

project contributed greatly in establishing the need for reconfigurable systems and in creating paradigms for parallel programming.

The personal workstation also opened up another market for the transputer, since it provides the basis for workstation accelerators. More than 50 companies now market



### The ESPRIT Supernode Project

Over about a four-year period (Dec. 1985-Nov. 1989), the Supernode Project aimed to create a low-cost, high-performance computer system from transputers. This project incorporated the development of the T800 floating-point transputer<sup>2</sup> and the Supernode RTP (Reconfigurable Transputer Processor).

One Supernode consists of 16 transputers connected via a crossbar switch (also developed during the project). Using crossbar switches, multiple Supernodes combine to provide systems of up to about 1,000 transputers with no limitations on topology (except those implied by the four links on each transputer). Constructing larger systems remains possible with minor constraints on topology.

The wide range of applications demonstrated on the Supernode included finite-element analysis, logic simulation, luminosity simulation, and real-time image analysis. Development of parallel algorithms and parallel numerical libraries also occurred.

The main results demonstrated the feasibility of the system, and the surprising ease in programming applications to take advantage of concurrency. A large number of European research projects base their development of parallel computing techniques on the now commercially exploited Supernode.

accelerator cards, offering a range of hardware and software capabilities. Initially these cards targeted the developing transputer embedded systems market. However, as application software for the transputer grew, the marketplace widened—first to engineering design workstations and more recently to commercial and financial workstations. Several workstations based entirely on transputers are now available.

ESPRIT projects continue to attack the difficult problems of providing very high performance systems (see box for current projects). The original issues concerned the feasibility of constructing such systems (possibly containing thousands of processors) and determining the methods of programming individual applications. The Supernode project developed a machine that could effectively execute a wide range of applications, and in many cases nearly achieved the theoretical maximum performance from the machine.

Current ESPRIT project issues involve "scalability," "portability," and programming ease. Scalability refers to the ease of achieving increased performance from an application by using more processors. Portability relates to the ease of transferring programs between parallel machines of different architectures. The term "general-purpose" summarizes these issues of parallel computing. A general-purpose par-

### Current ESPRIT projects

The Supernode 2 project studies software issues—particularly those concerned with advanced operating systems—in the context of the Supernode machine. Such an operating system converts the machine from one that runs one application into a more general-purpose facility.

The two-year PUMA (Parallel Universal Message-passing Architecture) project explores the possibilities offered by the new virtual communications architecture (see the "The next-generation transputer" section in the main article). Besides studying the performance offered by various topologies and the implications for computer architecture, the project examines high-level models of parallelism and their implementation using a virtual message-passing system.

The three-year GPMIMD (General Purpose, Multiple-Instruction, Multiple-Data) project aims to develop a standard European MIMD architecture based on H1 transputers and virtual communications. Four main European transputer-supercomputer manufacturers—Meiko, Parsys, Parsytec, and Telmat—currently work as key collaborators on the project, which is led by Inmos.

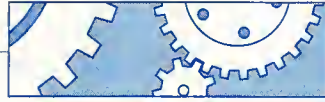
The OMI MAP (Open Microsystems Initiative Microprocessor Architecture Project) forms the core of the European-led Open Microsystems Initiative. It aims to define the architecture of a new microprocessor family designed to exploit the VLSI capabilities anticipated in the latter half of the 1990s. Collaborators include Bull, Inmos, Olivetti, Siemens, and Thomson.

Several other projects exploit the transputer architecture for specific application areas. For example, Padmavati (Parallel Associative Development Machine as a Vehicle for Artificial Intelligence) uses transputers and associative memory for symbolic processing.

allel computer executes a range of applications without concern for the number of processors employed or the topology in which they are connected. Indeed, whether the underlying hardware architecture is based on message passing (such as transputer-based and hypercube-style systems) or shared memory (such as Sequent's Balance system or Cedar systems) is of no concern to the programmer.

The expertise derived from ESPRIT and Supernode projects continues to spin off other applications. Migration of processing techniques and applications from general-purpose computers into special-purpose systems signifies a definite computing trend. For example, high-performance laser printers incorporating Postscript processing and intelligent terminals

*continued on p. 76*



# The European Declarative System, Database, and Languages

*The EP2025 EDS project develops a highly parallel information server that supports established high-value interfaces. We describe the motivation for the project, the architecture of the system, and the design and application of its database and language subsystems.*

Guy Haworth

Steve Leunig

ICL

Carsten Hammer

Siemens

Mike Reeve

ECRC

In 1988 Bull, ICL, Siemens, and their jointly owned European Computer Research Centre (ECRC) identified a common interest in supporting the processing of future intensive applications. The four partners defined a European Declarative System proposal, which the European Commission supported as project EP2025, a part of the European Economic Community's ESPRIT program. The EDS project began in 1989 and extends until 1992 with phases of definition, component development, and system integration.

EDS machines primarily function as information servers to manage all varieties of information intelligently. They will support languages and interfaces of value that are already established and in use: Unix, extended SQL, Lisp, C++, and the ECRC Elipsys parallel logic programming language.

The following analysis of the requirement for information servers motivated the EDS project and justifies this role for EDS machines.

## Information server requirement

Enterprises in the industrial, service, government, administrative, and defense sectors use information technology today. They depend increasingly on their information resource to:

- reduce operational costs,
- improve effectiveness from stock holding to customer service,

- support business development in new markets, and
- create a lasting competitive advantage in a rapidly changing world.

These enterprises often regard their information as more important than their next product or service. As a result, they pursue a systems architecture that delivers highly reliable information technology support and comprises a:

- complete, coherent, and robust information base;
- portfolio of applications interworking through data; and
- framework of long-life interfaces protecting their investments.

Large corporate systems increasingly play the role of database or information servers; servicing the SQL interface today requires some 75-80 percent of the processor cycles. Many factors increase the load on information servers, a fact likely to require the development of systems with highly parallel architecture to meet the future demand.<sup>1</sup>

**Information resource.** We've deliberately chosen the word *information* to be an umbrella term for the complete knowledge spectrum. We see this spectrum ranging from conventional formatted data to less structured text, representations of sound and image, and higher order knowledge in the forms, for example, of constraints, integrity



rules, business rules, and processes. Knowledge in its broadest sense, of course, includes facts and analysis, certainty, rumor, and speculation.

The volume of such information reportedly grows at some 25-30 percent yearly, a rate that we expect to be sustained by increased interest in text and image. Since this information is so valuable, we hope to store it with security levels that would be the envy of any bank. These levels suggest a large, central facility rather than a set of all too portable personal computer disks.

**Information access.** Only on-line systems support the effectiveness needed in our budget-conscious, competitive world. Literate information workers, desktop technology, CASE (computer-aided software engineering) tools, and business-to-business systems increase the volume of on-line transactions at some 20-30 percent a year. In addition, responses must come in a suitably short time, regardless of information volumes, transaction rates, or the incompleteness of the data input to specify the query.

We can characterize transactions in terms of frequency and complexity, as judged by the load they place on the computer system. Classic transaction processing occurs at a high rate with low complexity, while knowledge-based systems are highly complex and process at a low rate. Any computer system has a finite throughput capacity, shown by the performance frontier (see Figure 1).

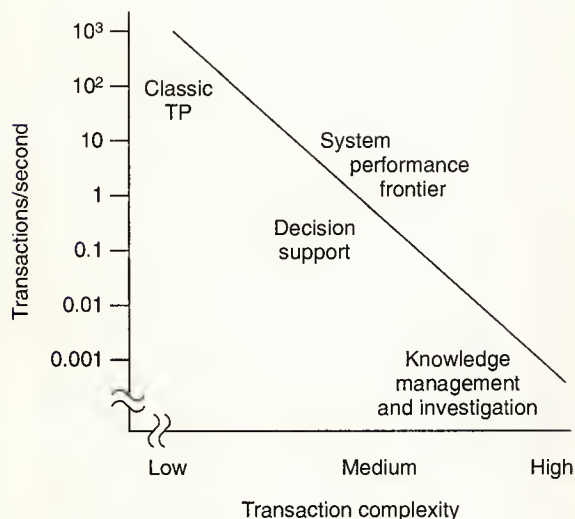


Figure 1. The range of transaction types.

Some evidence shows that performance problems impede the exploitation of "fifth-generation" knowledge-manipulation techniques. However, complex queries over knowledge bases do exist in several areas. These areas include govern-

ment administration, CAD (computer-aided design) systems including software engineering, the storage and scheduling system of distribution industries, and the remote maintenance systems of large utilities.

## EDS technology intercept

The EDS project aims to advance the information server performance frontier in the fastest way. We plan to do so by intercepting key hardware and software technologies and integrating them behind established interfaces of high value to prospective customers.

The ANSI/ISO SQL standard<sup>2</sup> is the key interface today between the application and the database manager. This query language always allows the user to ask for a set of records, most likely chosen from a large database. In principle, a million processors could simultaneously assess one each of a million records to service an SQL query with obvious benefits to the response time of the query.

Opinions differ as to whether SQL will evolve sufficiently to meet the new requirements for managing more complex data types and manipulating knowledge. We believe the current investment in SQL will guarantee SQL a long life. We therefore proposed an extension of SQL, ESQL, to meet future requirements for more comprehensive databases.<sup>3</sup>

On the hardware side, the most rapid change is occurring in microprocessors, which continue to increase in raw power at some 50 percent each year. Today, microprocessors promise 25 MIPS (million instructions per second); tomorrow, 40, 60, and 100 MIPS. The challenge for the computer system architect is to achieve a similar increase in total systems throughput.

In addition, storage technologies are diversifying; large-scale RAM storage is often the most cost-effective way to improve total systems performance and the price/performance ratio. We expect to see today's commodity, 4-Mbit dynamic RAM chip succeeded by the 16-Mbit chip in 1994 and the 64-Mbit chip in 1998. DRAM cost per byte now drops at 60 percent a year, and in 1995 we expect DRAM storage to be only 20 times the cost of magnetic-disk storage.

The EDS machine therefore exploits microprocessors and large DRAM storage, supported by a communications infrastructure of suitable responsiveness and bandwidth. It avoids the bottleneck of a single path from processing power to storage by adopting a distributed "share-nothing" architecture. This architecture offers linear performance returns when the number of processors increases into the hundreds.

## The EDS system

The static and simplified view of the EDS system seen in Figure 2 on the next page identifies the main interfaces and components. We designed the system to comprise a parallel

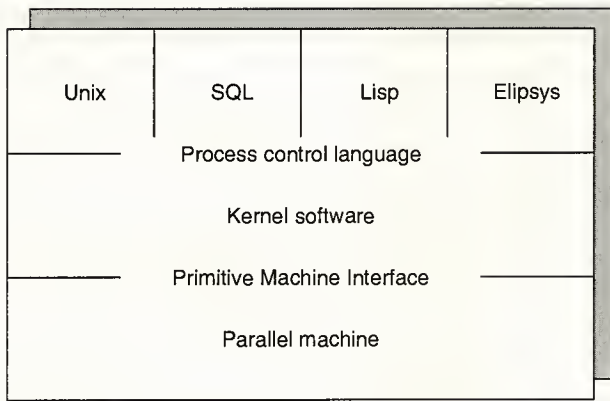


Figure 2. EDS system architecture.

processing machine and Emex kernel supporting Unix, extended SQL, Lisp, and Elipsys subsystems. The system will attach as an accelerator to a variety of Unix and proprietary hosts and be configurable up to 256 processors, each with up to 64 Mbytes of storage. We predict the following performance figures:

- *Database processing.* Meets the simple line-of-business Transaction Processing Council A benchmark performing 12,000 transactions per second at 30 percent utilization.
- *Lisp.* Meets the Boyer benchmark performing 140 Boyer runs per second.
- *Elipsys.* 32 MLIPS (million logical inferences per second) on average.

**The EDS hardware.** The EDS parallel machine<sup>4</sup> consists of a message-passing network, which provides a number of identical connection ports for attaching various functional elements. We envisage four types of elements: processing, diagnostic, input and output, and host connection (Figure 3).

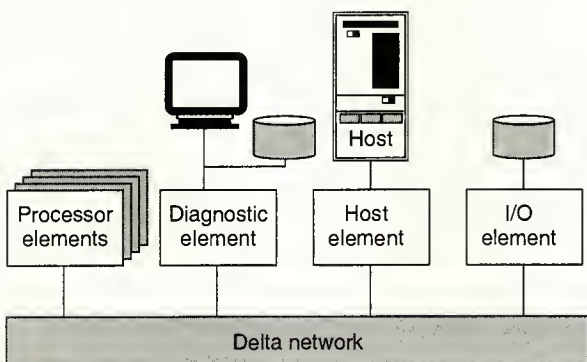


Figure 3. EDS hardware.

The processing element to be implemented for the prototypes consists of (see Figure 4):

- a main processing unit, a high-performance Sparc RISC (reduced instruction-set computer) with matching cache and memory management unit;
- a system support unit to offload the most critical parallelism primitives from the main processor;
- a network interface unit providing buffering and data transfer; and
- a local storage unit holding a maximum of 64 Mbytes of data.

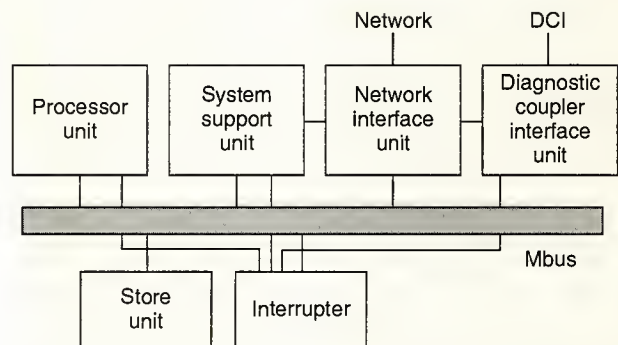


Figure 4. EDS processing element.

We expect later production versions of EDS to exploit the 16-Mbit chip and support 4 Gbytes of nonvolatile memory per processing element. We plan to simulate the effect of the well-understood nonvolatile storage during the project.

We designed the processing element to support efficiently the parallel operations of the execution models of the kernel, the database system, and the language systems. In addition to executing instructions within a normal sequential thread of computation, the processing element must support basic kernel operations such as passing a message. The primitive machine interface, or PMI, shown earlier in Figure 2, provides specific operations to support kernel functionality. PMI also introduces the required independence between the kernel software and the parallel machine hardware.

**The EDS kernel and PCL.** The EDS Process Control Language is the common interface through which all subsystems exploit and provide guidance to the parallelism features of the machine.

The concepts upon which PCL is based closely relate to those in Unix and in existing kernel interfaces such as Chorus Systeme's Chorus and Carnegie Mellon's Mach, both of which are designed to manage distributed systems. These concepts include virtual memory, processes, and interprocess communication. PCL develops these concepts to provide the



functionality and performance levels that a large-scale parallel system like EDS offers.

We particularly developed certain features of PCL in EDS:

- a multilevel process-context model with very lightweight threads,
- a storage model providing considerable flexibility in the sharing and management of virtual memory in a distributed system,
- efficient and reliable message passing,
- an exception-handling mechanism based on the message-passing scheme, and
- flexible scheduling and load balancing for a highly parallel system.

The inclusion within the EDS architecture of a common kernel and PCL interface brings a number of benefits: standard control of the machine, the exploitation of parallelism, and the use of system resources.

### The EDS database system

The main exploitation focus of the EDS project is the development of an advanced database server. The server provide an order-of-magnitude performance improvement over mainframes and advanced functionality to extend the range of applications it supports.

The improved functionality will include facilities for:

- support of user-defined data types and methods,
- support of complex objects and large objects,
- deductive database capabilities,
- general integrity constraints, and
- triggers (actions to be carried out when a given event occurs).

These features will not only extend the range of applications that can be supported efficiently and naturally but will also increase programmer productivity currently supported by standard relational database systems.

To achieve these objectives, we use a number of design strategies:

- exploitation of the parallelism available in the base EDS system;
- exploitation of large, stable RAMs to hold the persistent data over time and across system breaks;
- a database system based on standard relational database technology that is extended to provide object-oriented database and deductive database facilities;
- an interface, ESQL,<sup>3</sup> which is an extension of SQL. (The language provides a rich and extensible type system based on ADTs, or abstract data types, in which the

methods can be defined in various programming languages. It also provides complex objects with object sharing by combining the ADTs with object identity, and a Datalog-like deductive capability);

- database queries compiled into native machine instructions wherever possible; and
- an optimizer designed to be extensible to allow the system to evolve.

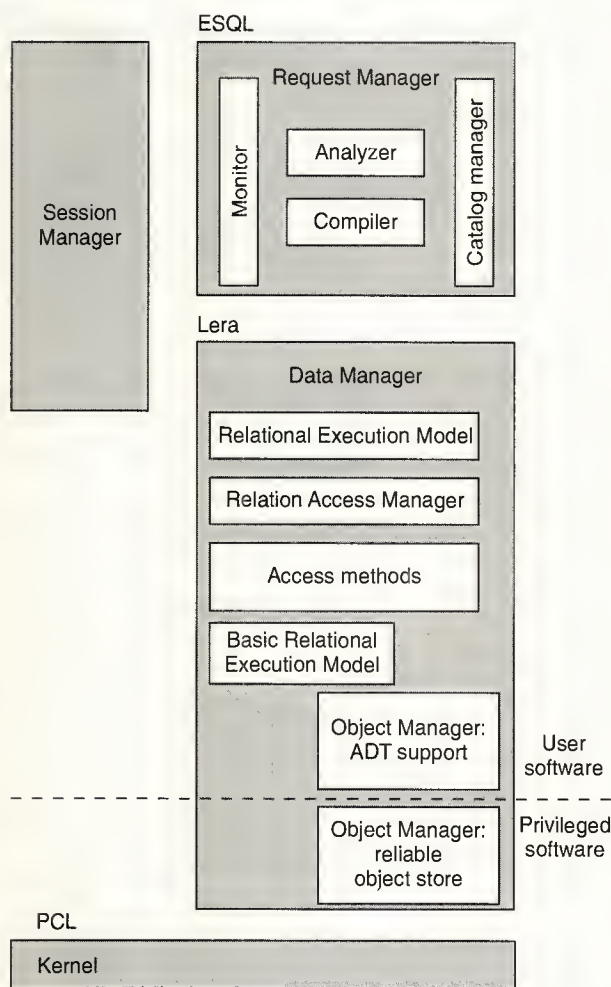


Figure 5. The logical structure of the database system.

**Database system architecture.** The database system splits into three main components, as shown in Figure 5. The Request Manager compiles database commands into a native machine code, the Data Manager provides the runtime facili-

*continued on p. 83*



# Architectures Within the ESPRIT SPAN Project

*The SPAN project pooled the resources of numerous researchers in several countries to integrate symbolic and numeric computing on parallel systems. The resulting Kernel System architecture provided a central model for which two programming languages and two parallel-system architectures were developed. This article sums up the project and discusses its advancements.*

Peter Rounce

University College

London

Jose Delgado

Instituto de Engenharia

de Sistemas e

Computadores

**E**SPRIT Project 1588, or SPAN (Symbolic Processing and Numeric), concerns the integration of numeric and symbolic computing on parallel architectures. The participants in this project are the Computer Technology Institute (Patras, Greece), the Instituto de Engenharia de Sistemas e Computadores (Lisbon), PCS Computer Systeme GmbH (Munich), Thomson-CSF Cimsa Sintra Division (Paris), Thorn-EMI Central Research Labs (London), the University of Athens, and the Department of Computer Science at University College London.

SPAN activities ranged from application software to VLSI (very large scale integration) hardware design and manufacture. The major goal of SPAN was to investigate symbolic and numeric integration on parallel computers at all levels: application, language, system, and architecture. This article briefly overviews the project before concentrating on its architectural research.

## The project

A three-year period of research completed January 1990 included:

- identifying parallelism in particularly complex applications and developing programs to demonstrate the use of this parallelism with parallel architectures,

- working at the language level to extend existing languages with parallel and other constructs to allow the development of programs for concurrent execution,
- working at the software system level to enable the easy mapping of concurrent programs onto existing machines, and
- working on novel architectures to provide increased hardware support.

We were aware of previous difficulties in programming parallel systems, porting existing software to them, exploiting the power of the systems, and extracting the full parallelism of the application. We wanted to develop an architectural model that would port to a number of parallel systems. This model would provide a fixed target for applications and languages, enabling easy porting of these elements to different hardware.

Most important, we wanted to investigate the integration of symbolic and numeric computing. Each type of computing is very powerful in its particular realm of application. Symbolic languages provide for concise and precise programs, by which complex programming tasks can often be simply expressed and proved formally correct. However, these languages are often slow to execute because their operational model does not map well to standard architectures.

Numeric languages, conversely, match standard



architectures particularly well and provide for fast execution and easy system control. The integration of symbolic and numeric computing in one environment promises to provide the powers of both with consequential results in programming capabilities.

We based the SPAN project on one architectural model of parallel computing. This model, called the Kernel System, is a generalized, abstract architecture. It directly supports both symbolic and numeric computing, as well as parallel processing. The Kernel System was the central component of the project to which all participants contributed, although Thorn-EMI and University College London had direct responsibility for the design and development. The Kernel System served as a target for the language and applications and facilitated their porting to standard architectures.

We started with two instantiations of the model.<sup>1</sup> Thorn-EMI was the primary developer of a high-level language called Parle (Parallel Architectures and Languages Europe).<sup>2</sup> UCL developed a low-level language called the Virtual Machine Code (VMC).<sup>3</sup>

We directed much of the work outside of developing the Kernel System toward introducing or applying parallelism to existing languages and applications:

- Thomson-CSF investigated Prolog and extended it with parallel constructs and numeric capabilities. This participant also developed a parallel numeric-symbolic system for image interpretation.
- PCS extended Lisp with parallel constructs.
- The Computer Technology Institute developed an expert system for the solution of partial differential equations, involving both symbolic and numerical processing in a single application. The investigations included how to identify and apply parallelism in the solution process.
- The University of Athens investigated the introduction of parallelism into a real-time database management system.
- Thorn-EMI designed a real-time expert system suitable for a parallel architecture.
- UCL developed an object-oriented environment for integrating symbolic and numeric programming.

The language work targeted Parle, the high-level interface to the Kernel System, while the applications targeted some subset of Prolog, Lisp, or Parle directly.

An associated activity concerned the development of an object-oriented programming environment. The principle aim was to investigate and develop an object-oriented framework for integrating heterogeneous hardware and software systems. Thus, the environment must provide for the production of programs that had components written in different languages. The environment must facilitate the exchange of data between the components. It would map

these programs onto diverse hardware systems—particularly parallel systems—and manage the execution of these programs. To some degree, this environment provided an alternative route to the integration of symbolic and numeric programming.<sup>4</sup>

Researchers at UCL developed such an environment, called Coside (originally called Solve). Further work is in progress.<sup>5</sup> Coside is a flexible system that can be targeted (more or less efficiently) at a large variety of systems and/or languages. The Kernel System is among them, and we expect it to provide efficient support for the environment.

We targeted a number of systems for the Kernel System at the architectural level. These included Supernode, developed by Thorn-EMI in ESPRIT Project 1058, and Padmavati (Parallel Associative Development Machine as a Vehicle for Artificial Intelligence), developed by Thomson-CSF in ESPRIT Project 967. The SPAN project produced three architectures: Sprint, DICE (Distributed Interconnected Computing Elements), and the D-machine. The objective of UCL's Sprint was to develop a parallel architecture to support the Kernel System efficiently. Sprint contains custom VLSI components and uses the VMC as the architectural model. (Figure 1 illustrates the relationship between projects within SPAN.)

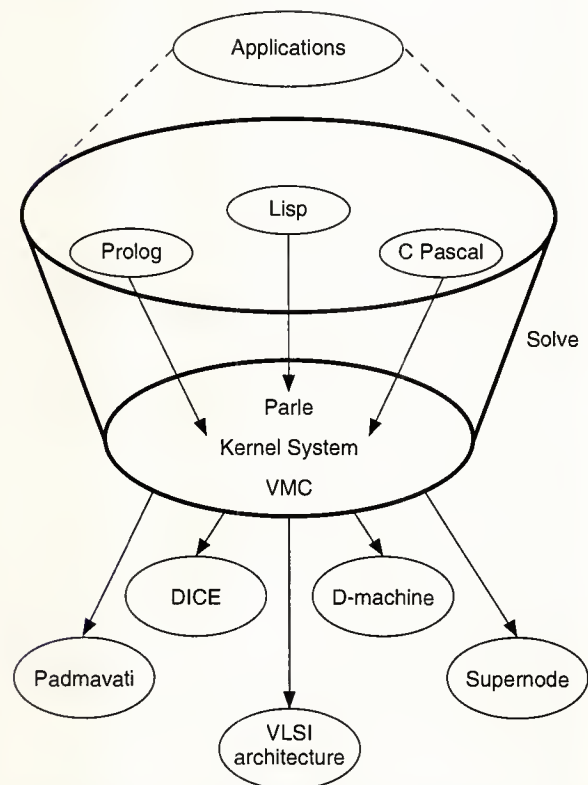


Figure 1. Organization of the SPAN project.

DICE is an object-oriented, parallel architecture developed by the Instituto de Engenharia de Sistemas e Computadores. It was initially targeted to be run on Inmos transputers. Inesc designed custom VLSI components to directly support the architecture.

The D-machine—a traditional bus-based multiprocessor developed from off-the-shelf components—uses Motorola 68030 processors. Its designer, PCS, also developed a real-time operating system for the D-machine. We present Sprint and DICE in greater detail later.

### Kernel System

This system provides an architecture that supports both symbolic and numeric computing with equal facility.<sup>1</sup> The Kernel System also provides multiple-instruction, multiple-data (MIMD) parallelism. The system contains key concepts taken from existing architectures for symbolic and numeric machines in an attempt to capture the best of both worlds. A key philosophy limited unnecessary complexity, providing only the essential primitives on which to develop complexity.

Although it is an abstract machine model, the Kernel System embodies a particular form of parallel architecture. Each processor has local, but not private, memory. The local memories form the logical, globally shared memory of the architecture. Any processor can communicate with any other processor via operations on the local memory of the remote processor, although the communications method is not specified. This approach provides a very general, shared-memory, MIMD architecture with point-to-point communications. The architecture can be mapped onto other styles, even though the communication costs might be greater and nonuniform in a real architecture.

In addition to shared memory, the system provides message-passing. A processor can use this procedure to access both its local memory and the nonlocal memories attached to other processors. The shared memory operations consist of the load and store functions common to many architectures in which a copy of a memory element is taken during a load operation and the current content of a memory element is overwritten upon a store.

For message passing, a memory element must have both "nonempty" and "empty" states. This arrangement allows a get operation to identify and remove the content of a nonempty element, leaving it empty. A put operation only writes a value into an empty element, making it nonempty. If the condition of an operation is not met, the system blocks operation and suspends the executing process until the condition is met.

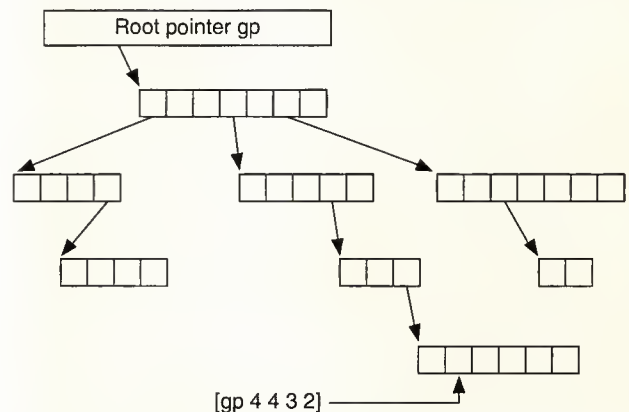
The two forms of operation provide the familiar asynchronous communication via shared memory and synchronous message-passing in a style similar to the Occam programming language. Synchronous message-passing is essential in a parallel processing environment, while shared

memory is a proven and powerful method.

Key features in the Kernel System support the integration of symbolic and numeric processing:

- the memory structure (through the provision of a memory element suitable for both styles of processing), and
- memory access mechanisms and data operators appropriate to both styles.

The list-structured memory of the Kernel System holds memory elements that are flexible in what they can contain. This memory structure can hold an integer or a list, or it can be empty. Lists of unlimited length contain all memory elements. Figure 2 shows a possible arrangement of the memory that is local to the processor. The root pointer gp points to the top-level list of the memory. This top-level list holds seven memory elements, three of which contain further lists. (The arrows in the figure indicate the lists held by particular memory elements.) Any list may contain other lists. Thus, the memory is tree structured and its depth is unlimited. The text [gp 4 4 3 2] in Figure 2 contains the address of the element to which the associated arrow points (see the Virtual Machine Code section in this article for further detail).





operators on the lists—supports symbolic processing.

The fact that the same lists are used for both styles of processing enables the integration of the two styles. The system can create a list in a symbolic way (like the cons operator of Lisp) and then process it in a numeric way. The list is then dismembered in a symbolic style (like the car and cdr operators of Lisp).

The processing model of the Kernel System incorporates both multiprocessing on individual processors and parallel processing on a set of processors. Each processor has its own code, which is stored in the list-structured, von Neumann memory. The execution model has procedural semantics with explicit flow of control. Each processor has a set of tasks or processes, which are scheduled in some arbitrary fashion.

## Parle

This high-level, procedural language follows the Algol style. It is a realization of the Kernel System. A key object in its design was supporting parallelism at both the processor and statement levels. Researchers initially envisaged Parle as a compiler target language. However, a number of the applications projects used Parle as a parallel programming language because it was much more developed at the beginning of the SPAN project than the other languages under investigation. This fact complicated the design and development of Parle, since in some instances the two goals were in conflict.

A Parle program consists of a set of data definitions and a set of processor definitions. The latter defines the code and associated local data to be loaded onto a processor in the target parallel system. Parle possesses a global memory that is accessed by name. Variables defined within a processor definition are local; others are global. Procedures, functions, variables, and processor definitions have names that allow them to be accessed and manipulated like other data. Since they are all implemented as lists, one would expect this capability.

The global name scheme provides for interprocessor communications. Parle can nest procedures, functions, and data inside other procedures and functions. Since Parle is a scoped language in which the scope of names is defined statically by the code definition, the system can hide data and methods.

A processor definition consists of local data, procedures and functions, and a code statement or statements. Some code is necessary for execution when the system starts up. All processors start up when code is loaded. This is the simplest and most general mechanism on which other startup mechanisms can be built. The mapping of code into processor definitions provides for the allocation of code to processors. The language provides replicators so that a processor definition can be mapped onto any number of processors.

Parle has operators for creating, joining, splitting, and selecting from lists:

- |                        |   |
|------------------------|---|
| 1) $a := [1, 2, 3, 4]$ | Create a list in variable $a$                                 |
| 2) $b := \# a$         | Take the length of the list in $a$                            |
| 3) $d := a ++ c$       | Join-concatenate two lists                                    |
| 4) $e := a << 2$       | Form sublist of $a$ from element 1 to and including element 2 |
| 5) $f := c << n$       | Form sublist of $c$ after element $n$ to end                  |
| 6) $f! 9 := a! 3$      | Copy element 3 of $a$ into element 9 of $f$                   |

Because Parle is loosely typed, a variable (memory element) can hold any data type—Boolean, integer, real, list, or empty. Therefore, the variables on the left-hand sides of the examples are untyped and accept whatever the right-hand sides produce. However, variables  $a$  and  $c$  in the last five examples must hold lists, since the length, join, split, and select operators must work on lists. Thus, the loose typing leads to a requirement for runtime type checking to determine the type of the contents of a variable.

The memory of Parle is list structured in the same way as the Kernel System. Copy semantics function as follows. Load operations always take a copy of the addressed memory, even if this means copying a list. In examples 3 through 5 above, the lists in  $a$  and  $c$  are unchanged by the operation. Lists can be deleted by assignment:

- |                              |                      |
|------------------------------|----------------------|
| $f := [5, 99, [1, 4, 6], a]$ | Create a list in $f$ |
| $f := 9$                     | Delete what's in $f$ |

These assignment statements demonstrate the shared memory operation of Parle. The variables may be local or global. Parle also provides for message-passing operations:

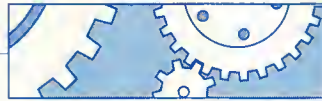
- |            |  |
|------------|--|
| $a := b?$  | Take a nonempty value from $b$ , leaving $b$ empty |
| $c? := a$  | Write into $c$ only when it is empty               |
| $d? := a?$ | Perform single-buffered communication              |

During these operations, the get operation leaves the read variable in the empty state. These operators block further processing, which only continues when the operations have succeeded in getting or putting a value.

Processor definitions allow for coarse-grain parallelism to be specified. Parle also supports fine-grained parallelism by providing operators for parallel execution at the statement level:

- i)  $s_1 || s_2 || \dots || s_n$
- ii)  $s_1 ; s_2 ; \dots ; s_n$
- iii) for  $\{i: 1 || n\} s_i \text{ rof}$
- iv) for  $\{i: 1 ; n\} s_i \text{ rof}$

*continued on p. 88*



# Pygmalion: ESPRIT II Project 2059, Neurocomputing

*Pygmalion aims to promote the European industry's application of neural networks and develop "standard" computational tools for their programming and simulation. A complete environment for developing algorithms and applications will demonstrate the network capabilities expected from their properties of massive parallelism, fault tolerance, adaptivity, and learning.*

Bernard Angeniol

## Mimetics

**I**n the last five years, we've seen a dramatic explosion of interest in neural computing that covered neural applications, models, programming environments, and neurocomputers. The Pygmalion project aims to promote the application of neural networks by European industry and to develop European "standard" computational tools for the programming and simulation of neural networks.

Tools for neural networks center on a programming environment comprising five major parts. They are a graphic monitor for controlling and monitoring a network simulation, an algorithm library of common neural networks, the high-level neural programming language N, the intermediate-level network specification language NC, and compilers for the target machines.

Pygmalion also addresses the implementation of neural algorithms by wafer scale integration techniques. In fact, a VLSI (very large scale integration) demonstrator for such a technology is already available. This development is the first step toward the building of a European general-purpose, highly parallel neurocomputer and the production of application-dedicated neurochips. These chips will be the goals of the future ESPRIT II project called Galatea.

Pygmalion applications span the fields of image processing, speech processing, and acoustic signals. We selected key real-world applications

in image processing and speech processing, and a small application in acoustic signals, to demonstrate the potential of neural networks to various industrial problems.

In image processing we investigated two important application domains, remote data sensing and factory inspection. Remote sensing includes pattern recognition and interpretation of Spot images on the earth's surface, such as road traffic, fields, and various kinds of grounds. Factory inspection covers the recognition and classification of workpieces in a factory automation context. These workpieces handle normal problems relating to position, overlap, and orientation, under different lighting conditions.

In speech processing, we planned to lay the foundations for an automatic speech recognition system by developing efficient learning algorithms for the basic building blocks. These basics include:

- isolated word recognition for small and medium-size vocabularies;
- speaker independence and adaptivity;
- speech preprocessing, including noise reduction;
- isolated word recognition in noisy environments; and
- subword-unit recognition and coarticulation.

Acoustic signal classification of underwater natural sounds applies neural computing in two



**Table 1. Pygmalion research groups.**

Partner	Laboratory	Role
Thomson-CSF	Division Systemes Electroniques, Paris	Prime contractor Image processing Acoustic signal processing N high-level language
INPG	Grenoble	VLSI neurocomputer chips
CSELT	Centro Studi e Lab. Telecomunicazioni, Torino	Isolated word recognition (IWR)
Philips	Lab. d'Electronique et de Physique appliquee, Paris	Image processing
SEL	SEL Research Centre, Stuttgart	Speaker-adaptive IWR
CTI	Computer Technology Institute, Patras	Cellular automata tools 3D pattern recognition
ENS	Ecole Normale Supérieure, Paris	Image pattern recognition
INESC	Instituto de Engenharia e Sistemas e Computadores, Lisboa	Algorithm library
IRIAC	Universite Paris Sud, Orsay	Algorithm library Low-level speech processing
UCL	University College London	Graphic monitor NC intermediate-level language
UPM	Universidad Politecnica de Madrid	Speech processing

different ways. It uses a processed version of the signal, obtained through classical preprocessing algorithms, as input to the neural classifier. It directly applies neural classification to the raw signal.

The Pygmalion project brings together many of the leading neural computing research groups from European industry, research institutes, and universities (see Table 1).

## NNPS

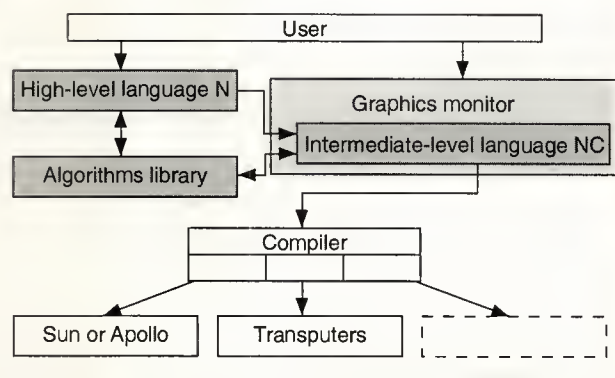
A major goal of Pygmalion is to ensure the widest usage of the neural programming environment by making it as flexible and portable as possible. The major parts of the Neural Network Programming System include the:

- *Graphics monitor*. The graphical software environment for controlling execution and monitoring of a neural network application simulation includes a simulation command language. This language sets up a simulation, monitors its execution, interactively changes values, and saves a trained network.
- *Algorithm library*. The parameterized library of common neural networks, written in the N language, provides users with a number of validated modules for constructing applications.
- *High-level language N*. This object-oriented programming language defines, in conjunction with the algorithm library, a neural network algorithm and application, by describing the network topology and its dynamics.
- *Intermediate-level language NC*. The low-level, machine-

independent network specification language represents the partially trained neural network applications, a format analogous to P code for Pascal systems.

- *Compilers*. Language is compiled for the target Unix-based workstations and parallel transputer-based machines.

Figure 1 illustrates this structure of the neural programming environment.



**Figure 1. Pygmalion neural programming environment.**

To ensure uniformity and consistency, the graphics monitor, the algorithm library, and the N and NC languages support a common interface with the following properties:

- 1) All components present the view of a neural system based on a hierarchical structure of networks, layers, clusters, neurons, and synapses.
- 2) All algorithms are parameterized.
- 3) All algorithms and applications share a common repertoire of data structures, function names, and system variables.
- 4) The specification of an algorithm is separated from an application.

The resulting uniform interface permits the development of generic applications and algorithms; one application may use many algorithms, and one parameterized algorithm may be configured for many applications.

**Graphics monitor.** This monitor controls a neural network simulation. It executes on a host computer, generating a specific net simulator/emulator to be executed on a target computer. The host, through X Windows graphics tools and the command language, monitors the simulation/emulation.

The windows environment provides pull-down menus to select and change the I/O format, network architecture, network learning algorithm, network training and execution, and displays of activations, weights, and so on.

Users initially specify a neural network in the N language using modules from the algorithm library. They then translate the program description into the NC language. Once configured and specified in NC, users can train or use the neural network via the monitor. The monitor lets them dynamically modify the network, translate it to a particular target machine, or save the trained or partially trained network for later usage, possibly on a different target computer.

**Algorithm library.** The library contains the classic neural network algorithms in a parameterized specification that can be configured for a specific user application. It includes the popular algorithms of:

- Gradient Back Propagation with or without feedback,
- Hopfield,
- Boltzmann Machine,
- Simulated annealing,
- Competitive learning,
- Adaptive resonance theory,
- Linear associative memory, and
- Kanerva memory.

These network algorithms already specify the interconnection geometry and the transfer equations. However, users select the number of processing elements, their initial state and weight values, learning rates, and time constants. The library divides into five main parts (see Figure 2):

- an algorithm-independent section that contains the network computation subroutines and support routines for memory and error management;
- tools library routines for data file I/O subroutines, network architecture specification, and performance measurement;
- algorithm modules, specifying the parameterized algorithms;
- algorithm evaluation programs for testing the algorithm modules; and
- the application library, which provides the user-tailored application modules.

We are constructing the algorithm library in two stages. Initially, we implemented a C version of the library, for use by the image and speech process-

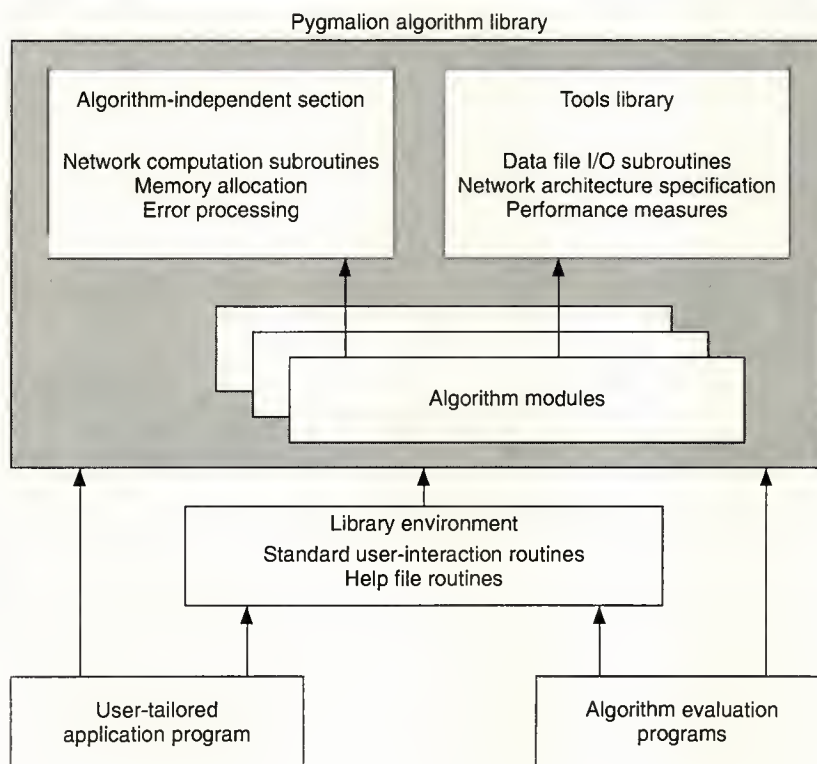


Figure 2. Pygmalion algorithm library.



ing applications in the project. Now, we are implementing the library using the N language, first with a reduced number of algorithms.

**The N language.** Both expert and naive users can use this high-level neural network programming language. They can develop neural algorithms and use operational algorithms in applications.

N's syntax is a subset of C++ with additional neural-oriented features. It allows the description of algorithms

- 1) by defining specific types having their own data and behavior in analogy to a class in C++ and
- 2) by assembling them in a modular tree hierarchy.

A library containing predefined types and parameterized algorithms (such as Hopfield, Back Propagation) accompanies N.

The library may also contain unprotected programming objects. Programmers manage this part of the library, which is used in a read-write mode. Storage of the source code of both kinds of library objects lets the objects be consulted and integrated easily into any N program.

A typical N program consists of a list of type definitions. One type may be defined from previously defined ones. Figure 3 displays the structure of a type definition.

```
new type xxx ( parameter-list )
  parameter declaration
  composite types declaration
  internal variables declaration
  internal function declaration
  above:
  inherited variables declarations
  plugin:
  communication fields (inputs)
  pluginout:
  communication fields (outputs)
  connection:
  connections between communication fields
  public:
  inheritance variables declaration
  activation methods of the type being defined
```

**Figure 3. Structure of a typical N program.**

Thanks to the choice made for its syntax, any program in N can easily be translated into a C++ program and consequently algorithms can be simulated on a sequential computer such as a Sun or Apollo. Furthermore, programmers can translate any neural network structure and algorithm in N into an equivalent NC structure, thus generating the nC version of any N program.

The N programming language provides three types of facilities in which programmers can:

- convert an N model (that is, algorithm) into an abstract representation that will allow the semantic analysis and the link editing of several algorithms in the same application;
- allow for the reuse of previously defined algorithms; and
- use tools and criteria to translate N applications into NC network specifications.

**The NC language.** This language acts as an intermediate-level, machine-independent representation for neural networks. Programmers can translate a network, specified in NC, onto a variety of computers for training or use. After usage (training), programmers update the intermediate-level specification of the network. The trained network can be stored in a library, filed for later use, or mapped onto a different computer. Machine independence is considered the major feature of the Pygmalion NC language, and, to enhance this, we have made the language a small subset of the draft ISO standard C.

The NC intermediate-level representation divides the neural network information into four different domains: the network topology, the data of the system including neuron status and synaptic weights, the functions defining the processing in the network, and the control of the network activities. An example of a framework for a neural network description appears in Figure 4 on page 99.

The topology information is basically described by defining the system variables (such as NETS, LAYERS, CYCLES, LEARNING\_RATE), using #define commands, and by completing the system and config structures. The system structure defines the central hierarchical structure in terms of nets, layers, and clusters inside layers. The config structure specifies the number of elements inside the hierarchy, such as the number of layers inside each network, number of neurons inside each cluster, and so on. The system structure also stores data information, in terms of a neuron's state and weights, as well as functional information in the form of rules. These rules relate to the functions that should be performed by a neuron, such as weight summation and weight update.

Two methods provide control information. Some system function definitions configure, initialize, and train the networks, while a list of calls to the system functions centrally control the whole system.

## Hardware integration

Although we dedicated only a very small amount of money to the study of hardware integration in Pygmalion, we made some progress. We studied a decentralized architecture in-

*continued on p. 99*

# IEEE Micro 1990 Index, Vol. 10

This index covers all technical items — papers, correspondence, reviews, etc. — that appeared in this periodical during 1990, and items from previous years that were commented upon or corrected in 1990.

The *Author Index* contains the primary entry for each item, listed under the first author's name, and cross-references from all coauthors. The *Subject Index* contains several entries for each item under appropriate subject headings, and subject cross-references.

It is always necessary to refer to the primary entry in the *Author Index* for the exact title, coauthors, and comments/corrections.

† means to check the main author's entry for subsequent corrections and comments.

+ means to check the main author's entry for coauthors.

## AUTHOR INDEX

### A

- Agrawal, Anant**, *see* Brown, Emil W., *M-M Feb 90* 10-22  
**Albertengo, Guido**, and Riccardo Sisto. Parallel CRC generation; *M-M Oct 90* 63-71  
**Alsop, Mitch**. Motorola's 88000 family architecture; *M-M Jun 90* 48-66  
**America, Pierre**, Ben Hulshof, Eddy Odijk, Frans Sijstermans, Rob van Twist, and Rogier Wester. Parallel computers for advanced information processing: A survey of ESPRIT Project 415; *M-M Dec 90* 12-15, 61-75  
**Angeniol, Bernard**. PYGMALION—ESPRIT 2 Project 2059—Neurocomputing; *M-M Dec 90* 28-31, 99-102  
**Asakura, Mikio**, *see* Hidaka, Hideto, *M-M Apr 90* 14-25

### B

- Banat, Karima**, *see* El-Imam, Yousif A., *M-M Aug 90* 62-74  
**Barnes, John**, *see* Birman, Mark, *M-M Feb 90* 55-64  
**Bier, Jeffrey C.**, Edwin E. Goei, Wai H. Ho, Philip D. Lapsley, Maureen P. O'Reilly, Gilbert C. Sih, and Edward A. Lee. Gabriel: A design environment for DSP; *M-M Oct 90* 28-45  
**Birman, Mark**, Allen Samuels, George Chu, Ting Chuk, Larry Hu, John McLeod, and John Barnes. Developing the WTL3170/3171 Sparc floating-point coprocessors; *M-M Feb 90* 55-64  
**Brown, Emil W.**, Anant Agrawal, Trevor Creary, Michael F. Klein, David Murata, and Joseph Petolino. Implementing Sparc in ECL; *M-M Feb 90* 10-22  
**Bural, David**, *see* Darley, Merrick, *M-M Jun 90* 36-47

### C

- Chamberlain, Roger D.**, and Mark A. Franklin. Hierarchical discrete-event simulation on hypercube architectures; *M-M Aug 90* 10-20  
**Chassaing, Rulph**, Wayne A. Peterson, and Darrell W. Horning. A TMS320C25-based multirate filter; *M-M Oct 90* 54-62  
**Cheang, S. M.**, *see* Lee, K. H., *M-M Aug 90* 50-61  
**Chu, George**, *see* Birman, Mark, *M-M Feb 90* 55-64  
**Chuk, Ting**, *see* Birman, Mark, *M-M Feb 90* 55-64  
**Churchill, Bob**, *see* Darley, Merrick, *M-M Jun 90* 36-47  
**Crawford, John H.** The i486 CPU: Executing instructions in one clock cycle; *M-M Feb 90* 27-36  
**Creary, Trevor**, *see* Brown, Emil W., *M-M Feb 90* 10-22

### D

- Darley, Merrick**, Bill Kronlage, David Bural, Bob Churchill, David Pulling, Paul Wang, Rick Iwamoto, and Larry Yang. The TMS390C602A floating-point coprocessor for Sparc systems; *M-M Jun 90* 36-47  
**Davis, Henry**, Robert Fine, and Denis Regimbal. Merging data converters and DSPs for mixed-signal processors; *M-M Oct 90* 17-27  
**de Lange, A. A. J.**, *see* van der Hoeven, A. J., *M-M Aug 90* 41-48  
**Delgado, J.**, *see* Rounce, P. A., *M-M Dec 90* 24-27, 88-97  
**Deprettere, E. F.**, *see* van der Hoeven, A. J., *M-M Aug 90* 41-48  
**Dewilde, P. M.**, *see* van der Hoeven, A. J., *M-M Aug 90* 41-48  
**Dyer, Stephen A.**, *Guest Ed.*, and Richard J. Higgins, *Guest Ed.* Introduction to special issue on the maturing of digital signal processing; *M-M Oct 90* 11-13

### E

- Edenfield, Robin W.**, Michael G. Gallup, William B. Ledbetter Jr., Ralph C. McGarity, Eric E. Quintana, and Russell A. Reininger. The 68040 processor—I: Design and implementation; *M-M Feb 90* 66-78  
**Edenfield, Robin W.**, Michael G. Gallup, William B. Ledbetter Jr., Ralph C. McGarity, Eric E. Quintana, and Russell A. Reininger. The 68040 processor—II: Memory design and chip verification; *M-M Jun 90* 22-35  
**El-Imam, Yousif A.**, and Karima Banat. Text-to-speech conversion on a personal computer; *M-M Aug 90* 62-74

### F

- Fine, Robert**, *see* Davis, Henry, *M-M Oct 90* 17-27  
**Franklin, Mark A.**, *see* Chamberlain, Roger D., *M-M Aug 90* 10-20  
**Fujishima, Kazuyasu**, *see* Hidaka, Hideto, *M-M Apr 90* 14-25

### G

- Gallup, Michael G.**, *see* Edenfield, Robin W., *M-M Feb 90* 66-78  
**Gallup, Michael G.**, *see* Edenfield, Robin W., *M-M Jun 90* 22-35  
**Goei, Edwin E.**, *see* Bier, Jeffrey C., *M-M Oct 90* 28-45  
**Govers, Francis X., III**, and Michael J. Pierson. On the Edge—Analysis of transactions and computers; *M-M Oct 90* 73-75

### H

- Hammer, Carsten**, Guy Haworth, Steve Leunig, and Mike Reeve. The European Declarative System, database and languages; *M-M Dec 90* 20-23, 83-88  
**Harrison, Beasley**. The Futurebus+ protocol stack and profiles (Ltr.); *M-M Jun 90* 2, 92-93  
**Hatano, Yuji**, Shinichiro Yano, Hiroyuki Mori, Hiroji Yamada, Mikio Hirano, and Ushio Kawabe. A 4-bit, 250-MIPS processor using Josephson technology; *M-M Apr 90* 40-55  
**Haworth, Guy**, *see* Hammer, Carsten, *M-M Dec 90* 20-23, 83-88  
**Hidaka, Hideto**, Yoshio Matsuda, Mikio Asakura, and Kazuyasu Fujishima. The cache DRAM architecture: A DRAM with an on-chip cache memory; *M-M Apr 90* 14-25  
**Higgins, Richard J.**, *Guest Ed.*, *see* Dyer, Stephen A., *Guest Ed.*, *M-M Oct 90* 11-13  
**Hirano, Mikio**, *see* Hatano, Yuji, *M-M Apr 90* 40-55  
**Ho, Wai H.**, *see* Bier, Jeffrey C., *M-M Oct 90* 28-45  
**Hootman, J.**, *Ed.-in-Chief*. Foreword to special issue on 1989 Annual Hot Chips Symposium; *M-M Feb 90* 2-3  
**Hootman, J.**, *Ed.-in-Chief*. The Far East issue for 1990 (Special issue foreword); *M-M Apr 90* 3  
**Hootman, J.**, *Ed.-in-Chief*. Hot chips, Part 2 (Special issue foreword); *M-M Jun 90* 3-4  
**Hootman, J.**, *Ed.-in-Chief*. Letters and articles (Edtl.); *M-M Aug 90* 2-3  
**Hootman, J.**, *Ed.-in-Chief*. How Micro survives (Edtl.); *M-M Oct 90* 3-4  
**Hootman, J.**, *Ed.-in-Chief*. Hellos and farewells; *M-M Dec 90* 3-4  
**Horning, Darrell W.**, *see* Chassaing, Rulph, *M-M Oct 90* 54-62  
**Hu, Larry**, *see* Birman, Mark, *M-M Feb 90* 55-64  
**Hulshof, Ben**, *see* America, Pierre, *M-M Dec 90* 12-15, 61-75

### I

- Iwamoto, Rick**, *see* Darley, Merrick, *M-M Jun 90* 36-47



## J

- James, David V.** Multiplexed buses: The endian wars continue; *M-M Jun 90* 9-21  
**Johnson, Stephen C.** Hot chips and soggy software; *M-M Feb 90* 23-26

## K

- Kadota, Hiroshi.** *see* Kaneko, Katsuyuki, *M-M Apr 90* 26-38  
**Kahaner, David K.** Software Report—Assignment Japan; *M-M Aug 90* 4-6  
**Kahaner, David K.** Software Report—The Pax parallel computer; *M-M Oct 90* 5-6, 91-93  
**Kahaner, David K.** Software Report—Quality improvement; *M-M Dec 90* 48-51  
**Kaneko, Hiroaki, Nariko Suzuki, Hiroshi Wabuka, and Koji Maemura.** Realizing the V80 and its system support functions; *M-M Apr 90* 56-69  
**Kaneko, Katsuyuki, Masaitu Nakajima, Yasuhiro Nakakura, Junji Nishikawa, Ichiro Okabayashi, and Hiroshi Kadota.** Processing element design for a parallel computer; *M-M Apr 90* 26-38  
**Kawabe, Ushio.** *see* Hatano, Yuji, *M-M Apr 90* 40-55  
**Kirrmann, Hubert.** Micro World—Cocom: The next wall to fall?; *M-M Feb 90* 4  
**Kirrmann, Hubert.** Micro World—Minitel: The French love affair with telematics; *M-M Apr 90* 88-90  
**Kirrmann, Hubert.** Micro World—Train control systems; *M-M Aug 90* 79-80  
**Kirrmann, Hubert.** Micro World—Train buses; *M-M Oct 90* 79-80  
**Kirrmann, Hubert.** Micro World—Reunification and the East German electronics industry; *M-M Dec 90*  
**Kitahara, Takeshi, and Taizo Satoh.** The Gmicro/300 32-bit microprocessor; *M-M Jun 90* 68-75  
**Klein, Michael F.** *see* Brown, Emil W., *M-M Feb 90* 10-22  
**Kronlage, Bill.** *see* Darley, Merrick, *M-M Jun 90* 36-47  
**Kumar, Krishna A., and Brian Petrasko.** Designing a custom DSP circuit using VHDL; *M-M Oct 90* 46-53

## L

- Lapsley, Philip D.** *see* Bier, Jeffrey C., *M-M Oct 90* 28-45  
**Ledbetter, William B., Jr.** *see* Edenfield, Robin W., *M-M Feb 90* 66-78  
**Ledbetter, William B., Jr.** *see* Edenfield, Robin W., *M-M Jun 90* 22-35  
**Lee, Edward A.** Programmable DSPs: A brief overview; *M-M Oct 90* 14-16  
**Lee, Edward A.** *see* Bier, Jeffrey C., *M-M Oct 90* 28-45  
**Lee, K. H., K. S. Leung, and S. M. Cheang.** A microprogrammable list processor for personal computers; *M-M Aug 90* 50-61  
**Leung, K. S.** *see* Lee, K. H., *M-M Aug 90* 50-61  
**Leunig, Steve.** *see* Hammer, Carsten, *M-M Dec 90* 20-23, 83-88  
**Lu, Mi.** *see* Sibai, F. N., *M-M Aug 90* 21-33  
**Luu, J.** Comments on 'A comparison of RISC architectures' by R. S. Piepho and W. S. Wu; *M-M Apr 90* 5 (Original paper, Aug 89 51-62)

## M

- Maemura, Koji.** *see* Kaneko, Hiroaki, *M-M Apr 90* 56-69  
**Mateosian, Richard.** Review of 'The Fifth Generation Fallacy' (Unger, J. M.; 1987); *M-M Feb 90* 7-8  
**Mateosian, Richard.** Review of 'Structured Walkthroughs, 4th edn.' (Yourdon, E.; 1989); *M-M Apr 90* 86-87  
**Mateosian, Richard.** Review of 'The Matrix—Computer Networks and Conferencing Systems Worldwide' (Quaterman, J. S.; 1990); *M-M Apr 90* 87  
**Mateosian, Richard.** Review of 'CASE—Using Software Development Tools' (Fisher, A. S.; 1988); *M-M Apr 90* 87  
**Mateosian, Richard.** Review of 'Computer Architecture—A Quantitative Approach' (Patterson, D. A., and Hennessy, J. L.; 1990); *M-M Jun 90* 5  
**Mateosian, Richard.** Review of 'Mastering Technical Writing' (Mancuso, J. C.; 1990); *M-M Oct 90* 76-77  
**Mateosian, Richard.** Review of 'Cache and Memory Hierarchy Design—A Performance-Directed Approach' (Przybylski, S. A.; 1990); *M-M Dec 90* 46-47

- Mateosian, Richard.** Review of 'The Elements of Spreadsheet Style' (Nevison, J. M.; 1987); *M-M Dec 90* 46-47  
**Matsuda, Yoshio.** *see* Hidaka, Hideto, *M-M Apr 90* 14-25  
**McGarity, Ralph C.** *see* Edenfield, Robin W., *M-M Feb 90* 66-78  
**McGarity, Ralph C.** *see* Edenfield, Robin W., *M-M Jun 90* 22-35  
**McLeod, John.** *see* Birman, Mark, *M-M Feb 90* 55-64  
**Milenkovic, Milan.** Microprocessor memory management units; *M-M Apr 90* 70-85  
**Miller, Christine.** Micro News—Silicon Glen: The European challenge; *M-M Jun 90* 7  
**Mori, Hiroyuki.** *see* Hatano, Yuji, *M-M Apr 90* 40-55  
**Murata, David.** *see* Brown, Emil W., *M-M Feb 90* 10-22

## N

- Nakajima, Masaitu.** *see* Kaneko, Katsuyuki, *M-M Apr 90* 26-38  
**Nakakura, Yasuhiro.** *see* Kaneko, Katsuyuki, *M-M Apr 90* 26-38  
**Nishikawa, Junji.** *see* Kaneko, Katsuyuki, *M-M Apr 90* 26-38

## O

- Odijk, Eddy.** *see* America, Pierre, *M-M Dec 90* 12-15, 61-75  
**Okabayashi, Ichiro.** *see* Kaneko, Katsuyuki, *M-M Apr 90* 26-38  
**Omnes, Jean-Francois.** *Guest Ed.*, Thierry Van der Pyl, and Philip Treleven, *Guest Eds.* Parallel computing in Europe; *M-M Dec 90* 8-10  
**O'Reilly, Maureen P.** *see* Bier, Jeffrey C., *M-M Oct 90* 28-45

## P

- Paterson, Tim.** On second thought... (Ltr.); *M-M Apr 90* 5  
**Pennello, Thomas J.** Compiler challenges with RISCs; *M-M Feb 90* 37-43  
**Peterson, Wayne A.** *see* Chassaing, Rulph, *M-M Oct 90* 54-62  
**Petolino, Joseph.** *see* Brown, Emil W., *M-M Feb 90* 10-22  
**Petrasko, Brian.** *see* Kumar, Krishna A., *M-M Oct 90* 46-53  
**Pierson, Michael J.** *see* Govers, Francis X., III, *M-M Oct 90* 73-75  
**Priem, Curtis R.** Developing the GX graphics accelerator architecture; *M-M Feb 90* 44-54  
**Pulling, David.** *see* Darley, Merrick, *M-M Jun 90* 36-47

## Q

- Quintana, Eric E.** *see* Edenfield, Robin W., *M-M Feb 90* 66-78  
**Quintana, Eric E.** *see* Edenfield, Robin W., *M-M Jun 90* 22-35

## R

- Reeve, Mike.** *see* Hammer, Carsten, *M-M Dec 90* 20-23, 83-88  
**Regimbal, Denis.** *see* Davis, Henry, *M-M Oct 90* 17-27  
**Reininger, Russell A.** *see* Edenfield, Robin W., *M-M Feb 90* 66-78  
**Reininger, Russell A.** *see* Edenfield, Robin W., *M-M Jun 90* 22-35  
**Rounce, P. A., and J. Delgado.** SPRINT and DICE: Architectures within the ESPRIT SPAN Project; *M-M Dec 90* 24-27, 88-97  
**Rumsey, Michael, and John Sackett.** An ASIC methodology for mixed analog-digital simulation; *M-M Aug 90* 34-40

## S

- Sackett, John.** *see* Rumsey, Michael, *M-M Aug 90* 34-40  
**Sakamura, Ken.** *Guest Ed.*, The current Japanese computer scene (Special issue intro.); *M-M Apr 90* 12  
**Samuels, Allen.** *see* Birman, Mark, *M-M Feb 90* 55-64  
**Satoh, Taizo.** *see* Kitahara, Takeshi, *M-M Jun 90* 68-75  
**Sibai, F. N., K. L. Watson, and Mi Lu.** A parallel unification machine; *M-M Aug 90* 21-33  
**Sih, Gilbert C.** *see* Bier, Jeffrey C., *M-M Oct 90* 28-45  
**Sijstermans, Frans.** *see* America, Pierre, *M-M Dec 90* 12-15, 61-75  
**Sisto, Riccardo.** *see* Albertengo, Guido, *M-M Oct 90* 63-71  
**Slater, Michael.** Micro View—The view from 10,000 feet; *M-M Feb 90* 96-95  
**Slater, Michael.** Micro View—Who needs faster processors?; *M-M Apr 90* 96, 95  
**Slater, Michael.** Micro View—What is RISC?; *M-M Jun 90* 96, 95  
**Slater, Michael.** Micro View—Failings of the patent system; *M-M Aug 90* 96, 95  
**Slater, Michael.** Micro View—Protecting computer architecture; *M-M Oct 90* 96, 95

- Slater, Michael.** Micro View—Phelps rules Intel breached contract with AMD; *M-M Dec 90* 96-95
- Stern, Richard H.** Micro Law—Appropriate and inappropriate legal protection of user interfaces and screen displays—V: How different forms of copyright protection interact with policy; *M-M Feb 90* 79-84
- Stern, Richard H.** Micro Law—Software patents; *M-M Apr 90* 8-11
- Stern, Richard H.** Micro Law—Professional ethics and the law; *M-M Jun 90* 83-84
- Stern, Richard H.** Micro Law—More on software patents; *M-M Aug 90* 7-9
- Stern, Richard H.** Micro Law—I: The Paperback case; *M-M Oct 90* 7-10
- Stern, Richard H.** Micro Law—The Paperback case—II: A 'nonliteral' analysis; *M-M Dec 90* 39-41
- Stock, S. J.** On the Edge—Low-cost CAD drawings; *M-M Aug 90* 77-78
- Suzuki, Nariko.** see Kaneko, Hiroaki, *M-M Apr 90* 56-69

T

- Treleaven, Philip.** *Guest Ed.*, see Omnes, Jean-Francois, *Guest Ed.*, *M-M Dec 90* 8-10

V

- van der Hoeven, A. J., A. A. J. de Lange, E. F. Deprettere, and P. M. Dewilde.** A model for the high-level description and simulation of VLSI networks; *M-M Aug 90* 41-48
- Van der Pyl, Thierry.** *Guest Ed.*, see Omnes, Jean-Francois, *Guest Ed.*, *M-M Dec 90* 8-10
- van Twist, Rob.** see America, Pierre, *M-M Dec 90* 12-15, 61-75

W

- Wabuka, Hiroshi.** see Kaneko, Hiroaki, *M-M Apr 90* 56-69
- Wang, Paul.** see Darley, Merrick, *M-M Jun 90* 36-47
- Warren, Carl.** Micro Standards—A busy year ahead; *M-M Feb 90* 85-86
- Warren, Carl.** On the Edge—Wire-to-wire interaction; *M-M Feb 90* 87-88
- Warren, Carl.** On the Edge—Realizing a transmission model; *M-M Jun 90* 76-79
- Warren, Carl.** Micro Standards—The scalable coherent interface; *M-M Jun 90* 80-82
- Warren, Carl.** Micro Standards—Backplane measurements; *M-M Aug 90* 75-77
- Warren, Carl.** Micro Standards—Keeping up with Uncle Sam; *M-M Oct 90* 72-73
- Warren, Carl.** Micro Standards—A performance standard to consider; *M-M Dec 90* 42-45
- Watson, K. L., see Sibai, F. N., M-M Aug 90 21-33**
- Wester, Rogier.** see America, Pierre, *M-M Dec 90* 12-15, 61-75
- Whitby-Stevens, Colin.** Transputers—Past, present and future; *M-M Dec 90* 16-19, 76-82

Y

- Yamada, Hiroji.** see Hatano, Yuji, *M-M Apr 90* 40-55
- Yang, Larry.** see Darley, Merrick, *M-M Jun 90* 36-47
- Yano, Shinichiro.** see Hatano, Yuji, *M-M Apr 90* 40-55

SUBJECT INDEX

A

- Analog-digital conversion**  
merging Sigma-Delta A/D converters and DSPs for mixed-signal processors. *Davis, Henry, +, M-M Oct 90* 17-27
- Application-specific integrated circuits**  
AMP (Analog Modeling Package), ASIC methodology for mixed analog-digital simulation. *Rumsey, Michael, +, M-M Aug 90* 34-40
- Arithmetic; cf. Floating-point arithmetic**
- Artificial intelligence**  
book review; The Fifth Generation Fallacy (Unger, J. M.; 1987). *Mateosian, Richard, M-M Feb 90* 7-8

B

- Bandpass filters**  
TMS320C25-based multirate filter. *Chassaing, Rulph, +, M-M Oct 90* 54-62

- Bipolar integrated circuits; cf. Emitter-coupled logic**
- Block coding; cf. Cyclic coding**

Book reviews

- Cache and Memory Hierarchy Design—A Performance-Directed Approach (Przybylski, S. A.; 1990). *Mateosian, Richard, M-M Dec 90* 46-47
- CASE—Using Software Development Tools (Fisher, A. S.; 1988). *Mateosian, Richard, M-M Apr 90* 87
- Computer Architecture—A Quantitative Approach (Patterson, D. A., and Hennessy, J. L.; 1990). *Mateosian, Richard, M-M Jun 90* 5
- Mastering Technical Writing (Mancuso, J. C.; 1990). *Mateosian, Richard, M-M Oct 90* 76-77
- Structured Walkthroughs, 4th edn. (Yourdon, E.; 1989). *Mateosian, Richard, M-M Apr 90* 86-87
- The Elements of Spreadsheet Style (Nevison, J. M.; 1987). *Mateosian, Richard, M-M Dec 90* 46-47
- The Fifth Generation Fallacy (Unger, J. M.; 1987). *Mateosian, Richard, M-M Feb 90* 7-8
- The Matrix—Computer Networks and Conferencing Systems Worldwide (Quaterman, J. S.; 1990). *Mateosian, Richard, M-M Apr 90* 87

- Business; cf. International trade**

C

Cache memories

- 68040 processor memory subsystem, external bus, chip and board testing, and design verification. *Edenfield, Robin W., +, M-M Jun 90* 22-35
- architecture of 88000 family of high-performance 32-bit microprocessors. *Alsop, Mitch, M-M Jun 90* 48-66
- book review; Cache and Memory Hierarchy Design—A Performance-Directed Approach (Przybylski, S. A.; 1990). *Mateosian, Richard, M-M Dec 90* 46-47
- cache DRAM, hierarchical RAM with 1-Mb DRAM main memory and 8-kb SRAM cache. *Hidaka, Hideto, +, M-M Apr 90* 14-25
- i486 CPU, 386-compatible processor with cache integrated into instruction pipeline. *Crawford, John H., M-M Feb 90* 27-36
- implementation of B5000 Sparc microprocessor in ECL. *Brown, Emil W., +, M-M Feb 90* 10-22
- the Gmicro/300 3d-bit microprocessor instruction execution, pipeline structure, and effect of internal caches. *Kitahara, Takeshi, +, M-M Jun 90* 68-75

- CAD (computer-aided design); cf. Design automation**

- CASE; cf. Computer-aided software engineering**

- Coding/decoding; cf. Cyclic coding**

- Communication systems; cf. Teletext/videtex**

Compilers

- crafting compilers for RISC processors. *Pennello, Thomas J., M-M Feb 90* 37-43

Computation time

- parallel unification machine for speeding up operation in execution of logic programs. *Sibai, F. N., +, M-M Aug 90* 21-33

- Computer-aided design; cf. Design automation**

Computer-aided software engineering

- book review; CASE—Using Software Development Tools (Fisher, A. S.; 1988). *Mateosian, Richard, M-M Apr 90* 87

Computer architecture

- book review; Computer Architecture—A Quantitative Approach (Patterson, D. A., and Hennessy, J. L.; 1990). *Mateosian, Richard, M-M Jun 90* 5
- comments on 'A comparison of RISC architectures' by R. S. Piepho and W. S. Wu. *Luu, J., M-M Apr 90* 5 (Original paper, Aug 89 51-62)
- parallel unification machine for speeding up operation in execution of logic programs. *Sibai, F. N., +, M-M Aug 90* 21-33
- protecting computer architectures legally (Micro View). *Slater, Michael, M-M Oct 90* 96, 95



**Computer buses; cf. Data buses**

**Computer communication; cf. Computer networks**

**Computer graphics**

GX graphics accelerator architecture. *Priem, Curtis R., M-M Feb 90 44-54*

**Computer input/output; cf. Computer interfaces**

**Computer instructions; cf. Microcomputer instructions**

**Computer interfaces**

appropriate and inappropriate legal protection of user interfaces and screen displays, part V (Micro Law). *Stern, Richard H., M-M Feb 90 79-84*

Lotus Development Corp. vs. Paperback Software International; court decision in copyright suit (Micro Law). *Stern, Richard H., M-M Dec 90 39-41*

Lotus Development Corp. vs. Paperback Software International; issues and decision in copyright dispute (Micro Law). *Stern, Richard H., M-M Oct 90 7-10*

personal-experience-based comments on legal protection of screen displays. *Paterson, Tim, M-M Apr 90 5*

**Computer interfaces; cf. Microcomputer interfaces**

**Computer language processors; cf. Compilers**

**Computer languages**

European Declarative System (EDS) database and languages. *Hammer, Carsten, +, M-M Dec 90 20-23, 83-88*

parallel computers for advanced information processing; survey of ESPRIT Project 415. *America, Pierre, +, M-M Dec 90 12-15, 61-75*

**Computer languages; cf. Hardware design languages**

**Computer networks**

book review; The Matrix—Computer Networks and Conferencing Systems Worldwide (Quateman, J. S.; 1990). *Mateosian, Richard, M-M Apr 90 87*

**Computer pipeline processing; cf. Parallel processing; Pipeline processing**

**Computers**

need for faster processors (Micro View). *Slater, Michael, M-M Apr 90 96, 95*

**Control systems; cf. Rail-transportation control systems**

**Copyright protection; cf. Software protection**

**Cyclic coding**

parallel encoding of cyclic redundant codes. *Albertengo, Guido, +, M-M Oct 90 63-71*

## D

**Data buses**

68040 processor memory subsystem, external bus, chip and board testing, and design verification. *Edenfield, Robin W., +, M-M Jun 90 22-35*

data-ordering issues for multiplexed buses. *James, David V., M-M Jun 90 9-21*

Futurebus+ protocol stack and profiles. *Harrison, Beasley, M-M Jun 90 2, 92-93*

operation of serial (wire) bus (On the Edge). *Warren, Carl, M-M Feb 90 87-88*

overview of IEEE draft standard P1194.O/D2, Backplane Electrical Performance Measurement (Micro Standards). *Warren, Carl, M-M Aug 90 75-77*

overview of proposed IEEE Standard P1596 for scalable coherent interface (Micro Standards). *Warren, Carl, M-M Jun 90 80-82*

**Data communication; cf. Computer networks**

**Data-flow computing**

designing a custom DSP circuit using VHDL. *Kumar, Krishna A., +, M-M Oct 90 46-53*

**Data processing; cf. List processing**

**Database systems**

European Declarative System (EDS) database and languages. *Hammer, Carsten, +, M-M Dec 90 20-23, 83-88*

**Delta modulation**

merging Sigma-Delta A/D converters and DSPs for mixed-signal processors. *Davis, Henry, +, M-M Oct 90 17-27*

**Design automation**

applicative state transition model for high-level description and simulation of VLSI networks. *van der Hoeven, A. J., +, M-M Aug 90 41-48*

**Design automation; cf. Design automation software**

**Design automation software**

AMP (Analog Modeling Package), ASIC methodology for mixed analog-digital simulation. *Rumsey, Michael, +, M-M Aug 90 34-40*

Gabriel, design environment for DSPs. *Bier, Jeffrey C., +, M-M Oct 90 28-45*

realizing a transmission model in SPICE (On the Edge). *Warren, Carl, M-M Jun 90 76-79*

software for low-cost CAD drawings on low-cost PC (On the Edge). *Stock, S. J., M-M Aug 90 77-78*

**Digital filters**

TMS320C25-based multirate filter. *Chassaing, Rulph, +, M-M Oct 90 54-62*

**Digital image processing; cf. Image processing**

**Digital integrated circuits; cf. Integrated circuits; Memories; Very-large-scale integration**

**Digital systems**

hierarchical discrete-even simulation on hypercube architectures; application to digital system simulation. *Chamberlain, Roger D., +, M-M Aug 90 10-20*

**Discrete Fourier transforms**

AMP (Analog Modeling Package), ASIC methodology for mixed analog-digital simulation. *Rumsey, Michael, +, M-M Aug 90 34-40*

**Discrete-time filters; cf. Digital filters**

**Displays**

appropriate and inappropriate legal protection of user interfaces and screen displays, part V (Micro Law). *Stern, Richard H., M-M Feb 90 79-84*

**Displays; cf. Computer graphics**

## E

**East Germany**

effect of reunification on East German electronics industry (Micro World). *Kirrmann, Hubert, M-M Dec 90*

**ECL; cf. Emitter-coupled logic**

**Electronics industry**

effect of reunification on East German electronics industry (Micro World). *Kirrmann, Hubert, M-M Dec 90*

implications of events in Scotland's Silicon Glen; interview with Robert Crawford. *Miller, Christine, M-M Jun 90 7*

**Emitter-coupled logic**

implementation of B5000 Sparc microprocessor in ECL. *Brown, Emil W., +, M-M Feb 90 10-22*

**Engineering profession**

professional ethics and the law (Micro Law). *Stern, Richard H., M-M Jun 90 83-84*

**Engineering writing; cf. Writing**

**Ethics; cf. Legal factors**

**Europe**

ESPRIT SPAN Project on integrating symbolic and numerical computing on parallel systems; overview and description of SPRINT and DICE architectures. *Rounce, P. A., +, M-M Dec 90 24-27, 88-97*

European Declarative System (EDS) database and languages. *Hammer, Carsten, +, M-M Dec 90 20-23, 83-88*

implications of events in Scotland's Silicon Glen; interview with Robert Crawford. *Miller, Christine, M-M Jun 90 7*

parallel computers for advanced information processing; survey of ESPRIT Project 415. *America, Pierre, +, M-M Dec 90 12-15, 61-75*

parallel computing in Europe (special issue). *M-M Dec 90 8-10*

PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angeniol, Bernard, M-M Dec 90 28-31, 99-102*

transputers review of European developments and applications. *Whitby-Stevens, Colin, M-M Dec 90 16-19, 76-82*

F

**Filters; cf. Bandpass filters; Digital filters; Sampled-data filters**

**Firmware; cf. Microprogramming**

**Floating-point arithmetic**

68040 processor design and implementation; operation of integer and

floating-point units. *Edenfield, Robin W.*, +, *M-M Feb 90* 66-78

TMS390C602A floating-point coprocessor for Sparc systems. *Darley, Merrick*, +, *M-M Jun 90* 36-47

WTL3170/3171 Sparc floating-point coprocessors; design and development. *Birman, Mark*, +, *M-M Feb 90* 55-64

**Fourier transforms; cf. Discrete Fourier transforms**

G

**Governmental activities/factors**

openness at standards meetings and US policy on sharing information (Micro Standards). *Warren, Carl*, *M-M Oct 90* 72-73

**Graphics; cf. Computer graphics**

H

**Hardware design languages**

applicative state transition model for high-level description and simulation of VLSI networks. *van der Hoeven, A. J.*, +, *M-M Aug 90* 41-48

designing a custom DSP circuit using VHDL. *Kumar, Krishna A.*, +, *M-M Oct 90* 46-53

**Hierarchical memories; cf. Memory hierarchies**

**Hierarchical systems**

cache DRAM, hierarchical RAM with 1-Mb DRAM main memory and 8-kb SRAM cache. *Hidaka, Hideto*, +, *M-M Apr 90* 14-25

hierarchical discrete-even simulation on hypercube architectures; application to digital system simulation. *Chamberlain, Roger D.*, +, *M-M Aug 90* 10-20

**History**

history and overview of programmable digital signal processes (DSPs). *Lee, Edward A.*, *M-M Oct 90* 14-16

**Home communication systems; cf. Teletext/videtex**

I

**IEEE standards**

Futurebus+ protocol stack and profiles. *Harrison, Beasley*, *M-M Jun 90* 2, 92-93

overview of IEEE Draft Standard P1194.O/D2, Backplane Electrical Performance Measurement (Micro Standards). *Warren, Carl*, *M-M Aug 90* 75-77

overview of proposed IEEE Standard P1596 for scalable coherent interface (Micro Standards). *Warren, Carl*, *M-M Jun 90* 80-82

**Image processing**

PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angniol, Bernard*, *M-M Dec 90* 28-31, 99-102

**Information systems; cf. Database systems**

**Integrated-circuit testing; cf. Microprocessor testing**

**Integrated circuits; cf. Application-specific integrated circuits; Very-large-scale integration**

**Integrated circuits industry; cf. Electronics industry**

**International trade**

impact of Cocom (Coordinating Committee on Multilateral Export Controls) restrictions (Micro World). *Kirrmann, Hubert*, *M-M Feb 90* 4

J

**Japan**

book review; The Fifth Generation Fallacy (Unger, J. M.; 1987). *Mateosian, Richard*, *M-M Feb 90* 7-8

innovative technology in the Far East (special issue). *M-M Apr 90* 14-69

report on visit to various university and nonuniversity research centers in Japan (Software Report). *Kahaner, David K.*, *M-M Aug 90* 4-6

**Josephson device logic**

4-bit, 250-MIPS processor using Josephson technology. *Hatano, Yuji*, +, *M-M Apr 90* 40-55

L

**Languages; cf. Computer languages**

**Laplace transforms**

AMP (Analog Modeling Package), ASIC methodology for mixed analog-digital simulation. *Runsey, Michael*, +, *M-M Aug 90* 34-40

**Legal factors**

appropriate and inappropriate legal protection of user interfaces and screen displays, part V (Micro Law). *Stern, Richard H.*, *M-M Feb 90* 79-84

court ruling that Intel breached contract with AMD (Micro View). *Slater, Michael*, *M-M Dec 90* 96-95

personal-experience-based comments on legal protection of screen displays. *Paterson, Tim*, *M-M Apr 90* 5

professional ethics and the law (Micro Law). *Stern, Richard H.*, *M-M Jun 90* 83-84

protecting computer architectures legally (Micro View). *Slater, Michael*, *M-M Oct 90* 96, 95

**Legal factors; cf. Patents; Software protection**

**List processing**

ASLP, PC-based highly pipelined low-cost microprogrammable list processor with two memory modules. *Lee, K. H.*, +, *M-M Aug 90* 50-61

**Logic circuits; cf. Emitter-coupled logic; Josephson device logic**

**Logic programming**

European Declarative System (EDS) database and languages. *Hammer, Carsten*, +, *M-M Dec 90* 20-23, 83-88

parallel unification machine for speeding up operation in execution of logic programs. *Sibai, F. N.*, +, *M-M Aug 90* 21-33

M

**Magnetic logic devices; cf. Josephson device logic**

**Meetings**

1989 (First) Annual Hot Chips Symposium, Part 1 (special issue). *M-M Feb 90* 10-78

1989 (First) Annual Hot Chips Symposium, Part 2 (special issue). *M-M Jun 90* 9-66

report on 1990 (2nd) Int. Workshop on Software Quality Improvement (Software Report). *Kahaner, David K.*, *M-M Dec 90* 48-51

**Memories; cf. Cache memories; Random-access memories; Virtual memories**

**Memory hierarchies**

book review; Cache and Memory Hierarchy Design—A Performance-Directed Approach (Przybylski, S. A.; 1990).

*Mateosian, Richard*, *M-M Dec 90* 46-47

**Memory management**

68040 processor memory subsystem, external bus, chip and board testing, and design verification. *Edenfield, Robin W.*, +, *M-M Jun 90* 22-35

architecture of 88000 family of high-performance 32-bit microprocessors. *Alsop, Mitch*, *M-M Jun 90* 48-66

overview of rationale, principles, and hardware support for microprocessor virtual memories. *Milenkovic, Milan*, *M-M Apr 90* 70-85

**Microcomputer architecture**

architecture of 88000 family of high-performance 32-bit microprocessors. *Alsop, Mitch*, *M-M Jun 90* 48-66

**Microcomputer instructions**

68040 processor design and implementation; operation of integer and floating-point units. *Edenfield, Robin W.*, +, *M-M Feb 90* 66-78

architecture of 88000 family of high-performance 32-bit microprocessors. *Alsop, Mitch*, *M-M Jun 90* 48-66

clarifying what is and is not RISC (Micro View). *Slater, Michael*, *M-M Jun 90* 96, 95

comments on 'A comparison of RISC architectures' by R. S. Piepho and W. S. Wu. *Luu, J.*, *M-M Apr 90* 5 (Original paper, Aug 89 51-62)

crafting compilers for RISC processors. *Pennello, Thomas J.*, *M-M Feb 90* 37-43

i486 CPU, 386-compatible processor with cache integrated into instruction pipeline. *Crawford, John H.*, *M-M Feb 90* 27-36



the Gmicro/300 3d-bit microprocessor instruction execution, pipeline structure, and effect of internal caches. *Kitahara, Takeshi*, +, *M-M Jun 90* 68-75

#### Microcomputer interfaces

overview of proposed IEEE Standard P1596 for scalable coherent interface (Micro Standards). *Warren, Carl*, *M-M Jun 90* 80-82

#### Microcomputer performance

standard approach to system performance measurement (Micro Standards). *Warren, Carl*, *M-M Dec 90* 42-45

#### Microcomputer software

Arabic text-to-speech conversion on a personal computer. *El-Imam, Yousif A.*, +, *M-M Aug 90* 62-74

Pax parallel computer family; overview of development and characteristics (Software Report). *Kahaner, David K.*, *M-M Oct 90* 5-6, 91-93

software for low-cost CAD drawings on low-cost PC (On the Edge). *Stock, S. J.*, *M-M Aug 90* 77-78

#### Microcomputer software design/development

crafting compilers for RISC processors. *Pennello, Thomas J.*, *M-M Feb 90* 37-43

eliminating software bottlenecks from chip design; case history of RISC development. *Johnson, Stephen C.*, *M-M Feb 90* 23-26

#### Microcomputer testing; cf. Microprocessor testing

##### Microprocessor testing

68040 processor memory subsystem, external bus, chip and board testing, and design verification. *Edenfield, Robin W.*, +, *M-M Jun 90* 22-35

#### Microprocessors

1989 (First) Annual Hot Chips Symposium, part 1 (special issue). *M-M Feb 90* 10-78

1989 (First) Annual Hot Chips Symposium, Part 2 (special issue). *M-M Jun 90* 9-66

32-b V80 microprocessor; internal hardware structure, pipeline operation, and system support functions. *Kaneko, Hiroaki*, +, *M-M Apr 90* 56-69

4-bit, 250-MIPS processor using Josephson technology. *Hatano, Yuji*, +, *M-M Apr 90* 40-55

68040 processor design and implementation; operation of integer and floating-point units. *Edenfield, Robin W.*, +, *M-M Feb 90* 66-78

68040 processor memory subsystem, external bus, chip and board testing, and design verification. *Edenfield, Robin W.*, +, *M-M Jun 90* 22-35

architecture of 88000 family of high-performance 32-bit microprocessors. *Alsup, Mitch*, *M-M Jun 90* 48-66

eliminating software bottlenecks from chip design; case history of RISC development. *Johnson, Stephen C.*, *M-M Feb 90* 23-26

i486 CPU, 386-compatible processor with cache integrated into instruction pipeline. *Crawford, John H.*, *M-M Feb 90* 27-36

implementation of B5000 Sparc microprocessor in ECL. *Brown, Emil W.*, +, *M-M Feb 90* 10-22

innovative technology in the Far East (special issue). *M-M Apr 90* 14-69

the Gmicro/300 3d-bit microprocessor instruction execution, pipeline structure, and effect of internal caches. *Kitahara, Takeshi*, +, *M-M Jun 90* 68-75

TMS320C25-based multirate filter. *Chassaing, Rulph*, +, *M-M Oct 90* 54-62

TMS390C602A floating-point coprocessor for Sparc systems. *Darley, Merrick*, +, *M-M Jun 90* 36-47

transputers review of European developments and applications. *Whitby-Strevens, Colin*, *M-M Dec 90* 16-19, 76-82

where microprocessors have been in past decade and where they will go in next (Micro View). *Slater, Michael*, *M-M Feb 90* 96-95

WTL3170/3171 Sparc floating-point coprocessors; design and development. *Birman, Mark*, +, *M-M Feb 90* 55-64

#### Microprogramming

ASLP, PC-based highly pipelined low-cost microprogrammable list processor with two memory modules. *Lee, K. H.*, +, *M-M Aug 90* 50-61

#### Multilevel systems; cf. Hierarchical systems

##### Multiplexing

data-ordering issues for multiplexed buses. *James, David V.*, *M-M Jun 90* 9-21

##### Multiprocessing

ESPRIT SPAN Project on integrating symbolic and numerical computing on parallel systems; overview and description of SPRINT and DICE architectures. *Rounce, P. A.*, +, *M-M Dec 90* 24-27, 88-97

European Declarative System (EDS) database and languages. *Hammer, Carsten*, +, *M-M Dec 90* 20-23, 83-88

hierarchical discrete-even simulation on hypercube architectures; application to digital system simulation. *Chamberlain, Roger D.*, +, *M-M Aug 90* 10-20

parallel computers for advanced information processing; survey of ESPRIT Project 415. *America, Pierre*, +, *M-M Dec 90* 12-15, 61-75

parallel computing in Europe (special issue). *M-M Dec 90* 8-10

parallel unification machine for speeding up operation in execution of logic programs. *Sibai, F. N.*, +, *M-M Aug 90* 21-33

Pax parallel computer family; overview of development and characteristics (Software Report). *Kahaner, David K.*, *M-M Oct 90* 5-6, 91-93

transputers review of European developments and applications. *Whitby-Strevens, Colin*, *M-M Dec 90* 16-19, 76-82

#### N

#### Networks; cf. Computer networks; Neural networks; Petri nets

##### Neural networks

PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angeniol, Bernard*, *M-M Dec 90* 28-31, 99-102

#### O

#### Object-oriented programming

parallel computers for advanced information processing; survey of ESPRIT Project 415. *America, Pierre*, +, *M-M Dec 90* 12-15, 61-75

#### P

#### Parallel processing

encoding of cyclic redundant codes. *Albertengo, Guido*, +, *M-M Oct 90* 63-71

VLSI-based processing element design for ADENA high-performance parallel computer. *Kaneko, Katsuyuki*, +, *M-M Apr 90* 26-38

#### Parallel processing; cf. Pipeline processing

##### Patents

benefits and problems of software patents (Micro Law). *Stern, Richard H.*, *M-M Apr 90* 8-11

examination of US Federal Circuit Court decision on software patent case involving EE issues (Micro Law). *Stern, Richard H.*, *M-M Aug 90* 7-9

failings of US patent system (Micro View). *Slater, Michael*, *M-M Aug 90* 96, 95

##### Petri nets

applicative state transition model for high-level description and simulation of VLSI networks. *van der Hoeven, A. J.*, +, *M-M Aug 90* 41-48

#### Pipeline processing

32-b V80 microprocessor; internal hardware structure, pipeline operation, and system support functions. *Kaneko, Hiroaki*, +, *M-M Apr 90* 56-69

68040 processor design and implementation; operation of integer and floating-point units. *Edenfield, Robin W.*, +, *M-M Feb 90* 66-78

ASLP, PC-based highly pipelined low-cost microprogrammable list processor with two memory modules. *Lee, K. H.*, +, *M-M Aug 90* 50-61

the Gmicro/300 3d-bit microprocessor instruction execution, pipeline structure, and effect of internal caches. *Kitahara, Takeshi*, +, *M-M Jun 90* 68-75

**Programming; cf. Software design/development**  
**Pulse-code modulation; cf. Delta modulation**

Q

**Quality control; cf. Software quality**  
**Quantization; cf. Analog-digital conversion**

R

**Rail transportation**  
 using microprocessors in trains and train buses (Micro World). *Kirrmann, Hubert, M-M Oct 90 79-80*  
**Rail-transportation control systems**  
 train control automation in Europe (Micro World). *Kirrmann, Hubert, M-M Aug 90 79-80*  
**RAM; cf. Random-access memories**  
**Random-access memories**  
 cache DRAM, hierarchical RAM with 1-Mb DRAM main memory and 8-kb SRAM cache. *Hidaka, Hideto, +, M-M Apr 90 14-25*  
**RD&E**  
 report on visit to various university and nonuniversity research centers in Japan (Software Report). *Kahaner, David K., M-M Aug 90 4-6*

S

**Sampled-data filters**  
 TMS320C25-based multirate filter. *Chassaing, Rulph, +, M-M Oct 90 54-62*  
**Semiconductor electronics industry; cf. Electronics industry**  
**Signal processing**  
 designing a custom DSP circuit using VHDL. *Kumar, Krishna A., +, M-M Oct 90 46-53*  
 Gabriel, design environment for DSPs. *Bier, Jeffrey C., +, M-M Oct 90 28-45*  
 history and overview of programmable digital signal processes (DSPs). *Lee, Edward A., M-M Oct 90 14-16*  
 maturing of digital signal processing (special issue). *M-M Oct 90 11-62*  
 merging Sigma-Delta A/D converters and DSPs for mixed-signal processors. *Davis, Henry, +, M-M Oct 90 17-27*  
 PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angeniol, Bernard, M-M Dec 90 28-31, 99-102*  
**Signal sampling/reconstruction; cf. Analog-digital conversion**  
**Simulation**  
 hierarchical discrete-even simulation on hypercube architectures; application to digital system simulation. *Chamberlain, Roger D., +, M-M Aug 90 10-20*  
**Software; cf. Computer languages; Design automation software; Microcomputer software**  
**Software design/development**  
 book review; The Elements of Spreadsheet Style (Nevison, J. M.; 1987). *Mateosian, Richard, M-M Dec 90*  
 report on visit to various university and nonuniversity research centers in Japan (Software Report). *Kahaner, David K., M-M Aug 90 4-6*  
 using transaction analysis to develop software for space station Freedom (On the Edge). *Govers, Francis X., III, +, M-M Oct 90 73-75*  
**Software design/development; cf. Microcomputer software design/development; Software development environments**  
**Software development environments**  
 PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angeniol, Bernard, M-M Dec 90 28-31, 99-102*  
**Software development management**  
 book review; Structured Walkthroughs, 4th edn. (Yourdon, E.; 1989). *Mateosian, Richard, M-M Apr 90 86-87*  
**Software protection**  
 benefits and problems of software patents (Micro Law). *Stern, Richard H., M-M Apr 90 8-11*  
 examination of US Federal Circuit Court decision on software patent case involving EE issues (Micro Law). *Stern, Richard H., M-M Aug 90 7-9*  
 Lotus Development Corp. vs. Paperback Software International; court decision in copyright suit (Micro Law). *Stern, Richard H., M-M Dec 90 39-41*

Lotus Development Corp. vs. Paperback Software International; issues and decision in copyright dispute (Micro Law). *Stern, Richard H., M-M Oct 90 7-10*

**Software quality**  
 report on 1990 (2nd) Int. Workshop on Software Quality Improvement (Software Report). *Kahaner, David K., M-M Dec 90 48-51*  
**Source coding; cf. Delta modulation**  
**Space stations**  
 using transaction analysis to develop software for space station Freedom (On the Edge). *Govers, Francis X., III, +, M-M Oct 90 73-75*  
**Special issues/sections**  
 1989 (First) Annual Hot Chips Symposium, Part 1. *M-M Feb 90 10-78*  
 1989 (First) Annual Hot Chips Symposium, Part 2. *M-M Jun 90 9-66*  
 innovative technology in the Far East. *M-M Apr 90 14-69*  
 maturing of digital signal processing. *M-M Oct 90 11-62*  
 parallel computing in Europe. *M-M Dec 90 8-10*

**Speech processing**  
 PYGMALION; survey of ESPRIT 2 Project 2059 on neurocomputing. *Angeniol, Bernard, M-M Dec 90 28-31, 99-102*

**Speech synthesis**  
 Arabic text-to-speech conversion on a personal computer. *El-Imam, Yousif A., +, M-M Aug 90 62-74*

**Standards**  
 openness at standards meetings and US policy on sharing information (Micro Standards). *Warren, Carl, M-M Oct 90 72-73*  
 system performance measurement; standard approach (Micro Standards). *Warren, Carl, M-M Dec 90 42-45*

**Standards; cf. IEEE standards**

T

**Teleconferencing**  
 book review; The Matrix—Computer Networks and Conferencing Systems Worldwide (Quaterman, J. S.; 1990). *Mateosian, Richard, M-M Apr 90 87*

**Teletext/videotex**  
 minitel, terminal for accessing French Teletel videotex service (Micro World). *Kirrmann, Hubert, M-M Apr 90 88-90*

**Text processing**  
 Arabic text-to-speech conversion on a personal computer. *El-Imam, Yousif A., +, M-M Aug 90 62-74*

**Trade; cf. International trade**

**Transforms; cf. Discrete Fourier transforms; Laplace transforms**

**Transmission lines**  
 realizing a transmission model in SPICE (On the Edge). *Warren, Carl, M-M Jun 90 76-79*

U

**United States; cf. Governmental activities/factors**

V

**Very-large-scale integration**  
 applicative state transition model for high-level description and simulation of VLSI networks. *van der Hoeven, A. J., +, M-M Aug 90 41-48*  
 VLSI-based processing element design for ADENA high-performance parallel computer. *Kaneko, Katsuyuki, +, M-M Apr 90 26-38*

**Videotex; cf. Teletext/videotex**

**Virtual memories**  
 overview of rationale, principles, and hardware support for microprocessor virtual memories. *Milenkovic, Milan, M-M Apr 90 70-85*

**VLSI; cf. Very-large-scale integration**

W

**Writing**  
 book review; Mastering Technical Writing (Mancuso, J. C.; 1990). *Mateosian, Richard, M-M Oct 90 76-77*





**Richard H. Stern**

Law Office of

Richard H. Stern

1300 19th Street NW,

Suite 400

Washington, DC 20036

## The *Paperback* case

### Part 2, A "nonliteral" analysis

**T**he court's opinion in *Lotus Development Corp. v. Paperback Software International*<sup>1</sup> did not approach the case in terms of statutorily listed, or even previously known, copyrightable subject matter. Instead, the court decided to devise a new species of copyrightable subject matter. It held that the "nonliteral" aspect of the Lotus 1-2-3 user interface that was embodied or reflected in its "command structure" was copyrighted.

The court defined the protected command structure as "the menu structure, taken as a whole—including the choice of command terms, the structure and order of those terms, their presentation on the screen, and the long prompts."

(The choice of command words means the collection of commands in the menu bars of 1-2-3. The structure and order of those terms means the tree relationship. That is the "tree" depicted in Figure 2 of Part 1. The presentation on the screen probably means the menu bars, although the court did not spell that

point out. The so-called long prompts are short explanatory messages that sometimes appear on the second line of a menu bar instead of a further list of commands. For example, under "Global," the explanation "Set worksheet settings" appears. The content of these prompts is minimal. It reflects both the simplicity of the message and the relatively few ways possible to state it. For all practical purposes, the gist of what the court decided to protect is the command set and the command tree.)

In the court's view, the nonliteral aspects of a computer program should be protected by copyright law. After all, the plot, characterization, and details of dialogue of a novel or play would be protected against nonverbatim copying. Why should a computer program be different? Basically, the court felt the nonliteral aspects of 1-2-3, specifically the user interface and command structure, were the most valuable part of the work as a whole. A major problem with this approach is that it is so unpredictable

### Highlights

Part 1 of this series described the most recent copyright decision in the screen display/user interface field, *Lotus Development Corp. v. Paperback Software International*. The decision created enormous concern in the software industry, because the court went out of its way to expound a broad theory that copyright law protects an unspecified collection of "nonliteral" aspects of computer programs.

Part 1 discussed the 1-2-3 computer program, command tree, and menu bars. It also explored the reason the defendant copied the command tree. The defendant claimed that the 1-2-3 input-command user interface had become an established convention in the spreadsheet field, a de facto standard.

The court chose not to analyze the issues in terms of the menu bars as conventional subject matter of copyright, such as pictorial works. It saw the case in a completely different light, to which Part 2 now turns.

in scope. It is without any coherent rationalizing principle, and totally destructive of business certainty.

The court's discussion, and the defendant's conduct challenged in the lawsuit, indicate the most important aspect of user interface and command structure protected by the *Paperback* decision. That aspect is the tree structure of the commands. Identically duplicating the tree was key to the defendant's attempt to achieve total compatibility. With such duplication, users could transfer their training and their macro libraries at minimal transaction cost.

Protecting the tree moves copyright protection of user interfaces to a previously unreached level of abstraction. All of the prior user-interface decisions appear to have decided whether there was copyright infringement on the basis of whether screen displays looked alike. (See the Prior Decisions box.) While that decision might have been reached in the *Paperback* decision, it was not. Instead, the court zeroed in on the reason why the parties' menu bars looked alike. The court protected the plaintiff's interest reflected in that reason as a nonliteral aspect of the underlying computer program. The reason, of course, was that both spreadsheet programs were being operated in accordance with the same input-command tree.

### The direct/indirect issue

A question was swept aside by the *Paperback* court's nonliteralist analysis that would or should have been suggested by a more literal approach. That question is whether it is appropriate to protect, by protecting something conventionally protected by copyright law, something else that does not itself fit within that or any other known category of things protected by copyright. Here, one might ask whether protecting input-command structure by protecting the menu bars as pictures is appropriate. Pictures are generally protectable, as such. But the plan is

### Prior decisions

No prior decision has expressly protected a command set and the structural, tree relationship of commands, as such. But a few prior decisions have some oblique intimations in this direction.

The concept of "structure" as a nonliteral, noncode element of a computer program derives from the decision in *Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc.*<sup>2</sup> In that case, the court held that the defendant infringed the copyright in the plaintiff's computer program, even though the codes were different. Indeed, the computer programs were written in dissimilar programming languages. The court based its conclusion of substantial similarity primarily on "structural" similarities of five subroutines and in the choice of data fields within certain types of files.

In *Manufacturers Technologies, Inc. v. Cams, Inc.*,<sup>3</sup> the court protected the "sequence and flow" of a set of menus as included within the copyright protection given to the computer program. The sequence and flow of the menus is a tree relationship similar to the tree relationship of the menu bars of the 1-2-3 program.

In *Digital Communications Assoc., Inc. v. Softklone Distrib. Corp.*,<sup>4</sup> the court found the main menu of the Mirror computer program to infringe the copyright in the main menu of Crosstalk. To some extent, although the court did not make a point of it, the court based its finding of substantial similarity on the fact that the same commands and parameters were listed in both menus under the same names and keystrokes. See *IEEE Micro*, Aug. 1989, p. 9.

really to protect the idea of these pictures—the command tree that they depict. Is it inappropriate to protect input-command structure indirectly when it cannot, under a literal approach, be protected directly? There is a division of authority on this point, with perhaps somewhat stronger support for the view opposed to protecting indirectly that which is unprotectable directly.

The US Supreme Court's decision in *Baker v. Selden*<sup>5</sup> suggests one view. The Supreme Court held that one cannot protect bookkeeping forms that are a "necessary incident" of an art taught in a book on bookkeeping. The court reasoned that the art of bookkeeping is not protectable as such by copyright. Thus, it would be improper to protect the art indirectly by protecting directly the forms needed to practice the art. Nowadays, courts refer to this reasoning as the merger of idea and expression.

On the other hand, courts have not protected houses by copyright, although copyright law protects drawings (including those of houses) against reproduction. Yet, as a practical matter one cannot efficiently copy a house without copying the blueprints for the house. Courts also hold that copying such blueprints is copyright infringement.<sup>6</sup> Thus, copyright law may indirectly protect the house against copying by protecting the blueprints.

Under the *Paperback* approach, considering the question whether it is inappropriate to protect the command tree indirectly is unnecessary. The court simply protects it directly by defining it as a nonliteral aspect of the copyrighted computer program. That is one way to avoid a troublesome problem.

### Functionality

Several aspects of the *Paperback* court's treatment of possibly functional aspects of the 1-2-3 command tree deserve mention. Unquestioned copyright doctrine holds that copyright law does not protect functional aspects of com-



puter programs or other works. However, often disagreement arises over what aspects of a computer program are functional.

In the Micro Law series on screen displays and other user interfaces,<sup>7</sup> I suggested a solution to the disagreement. A proper concept of functionality includes whatever makes a computer program easier and faster to learn and use, leads to fewer user errors, or is otherwise better or cheaper than alternatives. This concept included conventions (such as using <F1> to invoke Help menus). It also included standards, whether de facto (such as the QWERTY keyboard) or *de jure* (either blessed by the IEEE or ANSI or commanded by a governmental body).

In the *Paperback* decision, the court emphatically and comprehensively rejected any aspect of convention, de facto standard, or commercial need for compatibility as a justification for use of the command tree. First, the court denied that compatibility with 1-2-3 was commercially necessary. To show it was not, the court pointed to the existence of other spreadsheets on the market that are not compatible with 1-2-3. Indeed some of them have had superior functionality, performing tasks that 1-2-3 did not. But the court did not address the market position of the competitive products.

It appears that no competitor has been able to gain a significant market share, and that 1-2-3 remains dominant. That fact, unremarked by the court, would seem to undercut the materiality of the mere existence of competition. (If incompatible spreadsheets are only marginally viable, it would seem highly material to determining whether a situation like that of the QWERTY keyboard exists.) At the very least, the matter should have been addressed. Instead, the court made a conclusory assertion that compatibility is commercially unnecessary.

Second, the court said that the defendant should have dealt with the compatibility problem, if one existed.

The defendant, the court said, should have written Help menus to explain to users how to substitute different keystrokes for those of 1-2-3. Also, the defendant should have written a computer program to convert customers' macros from the existing 1-2-3 command format to another, new format.

The first part of this suggestion just ignores commercial practicability. To the extent that simple 1:1 keystroke substitutions are involved, the suggestion is comparable to giving a typist a Dvorak keyboard in place of the QWERTY keyboard, along with a chart showing the substitutions. To the extent that the command tree or the basic logical approach is to be changed, the suggestion is like sending the average user to Bulgaria with nothing but a pocket English-Bulgarian phrase book and the court's good wishes.

As for the macro translator, the court said that it was a commercially practicable solution because another spreadsheet vendor had adopted it. Again, the court failed to address the acceptance of this expedient in the marketplace, as measured by market share. If, as appears to be the case, the market share of the company using the macro translator is very small compared to 1-2-3's share, this is probably not a commercially practicable proposal.

The court's proposals are remarkably unhelpful. You can think of it this way. Say the defendant wanted to market an English-language computer program. Then, it claimed that it had to put the subject of a statement or sentence first, then the verb, and then the object, as in "The dog bit the boy." Whereupon the defendant might receive this response:

You may as well say, "The dog boy bit," and just assume that you are speaking German. Your customers will probably get used to it, eventually. And you can give them a Help chart explaining how to speak in German instead of in English.

Or, as long as you are providing a Help chart, why don't you just add case endings and declensions and that sort of thing? Then you can scramble the order randomly and tell the customers to assume that they are speaking in Latin.

And while you are at it, don't feel limited to Indo-European languages. Chinese has some perfectly fascinating possibilities for icons and ideograms. Don't worry, your customers may well adjust to it.

In the next issue, I will continue this analysis with a short discussion of the standardization aspects of functionality. Afterward, I will comment about what is going on in the *Paperback* case, and more generally in software cases.

---

## References

1. 15 U.S.P.Q. 2d 1577 (D. Mass. 1990).
2. 797 F.2d 1222 (3d Cir. 1986), *cert. denied*, 107 S. Ct. 877 (1987).
3. 706 F. Supp. 984 (D. Conn. 1989).
4. 659 F. Supp. 449 (N.D. Ga. 1987).
5. 101 U.S. 99 (1879).
6. *Imperial Home Corp. v. Lamont*, 458 F.2d 895 (5th Cir. 1972).
7. *IEEE Micro*, June 1989 to Feb. 1990.

---

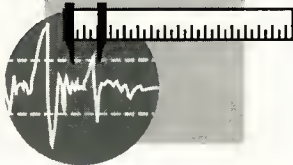
## Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 171    Medium 172    High 173

---

# Micro Standards



## A standard to consider

**S**ystem designers face a significant problem: What is the best way to measure system performance? Clearly, a few simple benchmarks aren't the answer. We need to develop a standard way of measuring system performance. I propose an approach that I believe is workable. I hope you will add your own ideas; possibly a formal standards working group will form.

Some of the ideas presented here come from the concept of rate monotonic scheduling—a statistical method of determining CPU usage. You may want to consult the work proposed by Jerome C. Huck and Michael J. Flynn in an excellent IEEE Computer Society book, *Analyzing Computer Architectures* (\$42 nonmembers, \$31.50 members).<sup>1</sup> This book provides a foundation for developing a system-measurement method. I use the authors' Canonic Interpretive (CI) method as a basis for measuring the effects of a high-level language (see the Proposed Terms box). In this case the language is Ada, developed by the US Department of Defense.

The CI method of measurement assesses the size of the smallest representation of a high-level-language program in relationship to the system resources (CPU, memory). CI also helps to determine how much work must be done by the system to make use of the program code.

For example, let's consider a system that serves as a multitasking unit. Say that power, budget, and environment considerations constrain the memory. The system operates with all CSCI (Computer Software Configuration Item)<sup>2</sup> components in memory at one time (the sizes are questionable). Its equivalency factor is based

on KIPS, or thousands of instructions per second, a term specific to small-scale processors.

Using the KIPS of a system related to source lines of code is probably an invalid approach when applied to most processor environments. It doesn't account for compiler capability or compactness. The KIPS formula says that for every source line of code,  $n$  machine instructions execute. This concept would be interesting if it were true—but it isn't. Further, the KIPS approach describes an interpretative method. In other words, each line of code must be interpreted, from high-level Ada to machine code. This is simply not the case.

Our example also assumes that the CPU power is directly equitable to a MIPS (million instructions per second) rating.

Although a MIPS rating can give you a rough idea of how much CPU horsepower is available, it doesn't provide you with a true picture of the system. To determine this, you should evaluate the appropriateness of the machine in relationship to your subsystem requirements. Process speed, throughput, and availability of memory are all important. You should also establish a deterministic model that is useful in all cases for all processors. It's also important to limit the application control in relationship to container (memory) size.

The KIPS approach also fails to account for several performance factors (especially in terms of Intel Corp.'s 386 microprocessor). These include source-code compilation, processing and memory architecture, hardware/software considerations; and operating system processing.

The 386 compiles source code (in the best cases by an optimizing compiler). The source

*continued on p. 44*

Carl Warren

McDonnell Douglas

Space Systems

(714) 896-3311

x. 6-0669

MCI mail: 310-9380;

Compmail: c.warren



## Proposed terms

*Activity* refers to the total number of CI objects accessed. Try to keep this number small; remember small models work better than large ones.

*Block size* is the amount of memory needed to store a defined environment. I use 1 Mbyte, although this amount can be reduced for a transfer function that involves 1-Kbyte blocks, for instance.

*Block work* is a measure of the CPU time needed to process a block of data.

The *Canonic Interpretive* (CI) method statistically measures a process and actually forms the basis for monotonic rate scheduling. CI operations roughly equate to instructions. CI forms a lower bound so you can assume an "ideal" boundary, and it has no knowledge of frequency distribution of program constructs, static or dynamic. Therefore, this measure considers any compiler optimization to be complete and assumes the source to be in its best form. ("Best form" means that care was taken in creating well-crafted code.)

A *container* describes the boundaries of the environments. In this case, a container size may be 4 Mbytes. This container is defined as either a 16-bit or a 32-bit word, one of the elements that the standard should determine. The word size determines how I/O is measured.

*Control transfer* points label a point at which the procedure transfers control, data, or function to another procedure (forward or backward).

The *correspondent* assumes a one-to-one relationship with executed instructions and operands. Therefore, the number of CI operations (instructions) executed equals the dynamic number of high-level-language actions in the source program. Thus, a procedure made up of Fors and Tos can be treated as one high-level action.

*CPU intensity* refers to block size and block work. The goal is to approach unity, which would denote one processor cycle per word. The best you can presently achieve is about five cycles per word.

*CPU utilization* concerns a percentage measurement of the CPU time/CPU time + I/O time.

*Distance* is a measurement of the space between objects in transition times. This measurement is only important when a large number of similar transition points exist. You may even use this for "virtual" objects. So {An} may be *n*-transition (machine-cycle) times away from {Bn}.

*Environment* refers to the bounds of the program region (you can call this a procedure, or a CSCI).

A *Gibson Mix* is the totality of typical, executed instructions of a processor instruction set. For example, while the IBM 370 has 378 instruction types, only 78 are used.

*J* (elapsed time) is the average time necessary to transfer 1 Mbyte of data via the I/O system. (You need to know the characteristics of the I/O.)

*K* (processing time) is the CPU time needed to process a 1-Mbyte block.

The statistical *Kendall ranking*<sup>3</sup> method doesn't assign an artificial index. It ranks positions of procedures to other elements in a system, as opposed to a Spearman ranking, which is monotonic. To determine the right Kendall ranking, you analyze the collection of procedures and give them a name of your choice (for example, 1 or 2). The method lets you choose the base and bounds. This ranking is by nature bounded, or it has no meaning.

A *MIMD* (multiple-instruction/multiple-data) stream defines the way a processor fetches instructions and data from logical memory. Thus a processor fetches or pipelines more than one instruction and associated data into its registers. The 386 operates this way to keep the fetch queues full and maximize processing time. A 20-MHz 386 takes 10 machine cycles to perform a memory-to-register move instruction. This common test is one of the more prevalent activities that a processor performs.

The *objects* notion can be confusing because it has several meanings, depending on what you are doing. For our purposes, it means operations, operands, and control/transfer points.

*Procedure* has a complex definition. A procedure consists of code elements with a beginning, a middle, and an end. A procedure can be one statement.

A *SIMD* (single-instruction/multiple-data) stream characterizes typical von Neumann processors.

*Size* refers to the extent of each object under consideration—in bits. The system equation is  $\text{Log}_2$  (number of similar objects in the environment).

*Stability* refers to the notion that the number of environment and control transfer points are kept to a minimum. In other words, you should try to find the point of stability at which you get the most meaningful information.

A *value machine* is an idealized model of the physical machine. No perturbations are allowed that may upset the operation of the machine, at least for the analysis.

*Work load* refers precisely to what the system can withstand in terms of memory usage, I/O, and performance.

code elements match the instruction set of the processor. Therefore, just a few instructions can represent several lines of code. For example, the processor treats active standing verbs, such as Begin and End, and library functions once—not multiple times. The Data Analysis Center at Wright-Patterson Air Force Base denotes a typical translation as 1.8 to 2.4 instructions per line of Ada source code, based on many optimized compilations.

The 386 processor and its memory architecture are robust. The 386 controls the memory and its segmentation. Further, the 386 allows you to lock out (protect) segments of memory based on boundaries. In addition, the 386 provides multiple hardware stacks and supports software stacks that are independent of the hardware.

The processing environment defines itself based on the operating system. The OS scheduler, working in tandem with the 386 processor, determines what code is resident at any time.

Another 386 factor is its internal Harvard architecture, which can fetch data and instructions in parallel. This fact makes a big difference in deciding how to create the proper model.

### Definitions

I proposed the earlier set of terms to serve as part of this standard. I took some of them from Huck and Flynn; others are my own. (I don't discuss all the terms listed, making the assumption that readers have a fundamental knowledge of system analysis.)

### Analysis

The whole purpose of analyzing a system is to measure the work load in a value machine. This process determines the necessary cache and memory requirements, and establishes some notion of the I/O parametrics.

CI establishes the number of operations, operands, and labels necessary to create a program or CSCI. To make this analysis work, we need to

determine CI. Functions of operations, operands, and labels appear as syllables. The syllable is an important determination and therefore must be uniform. Using KIPS based on source lines of code is only accurate for an interpretive environment. Here, we'll ignore the frequency and introduce some number we can establish for conversion.

For our purposes, I chose the number ranges from 1.8 to 2.4 provided by the Data Analysis Center rather than the number 7 as selected in Huck and Flynn. You can always increase the number if you need to consider swag in the conversion of source code to executable objects.

See the table for a system profile based on CI measuring methods. A Kendall ranking<sup>3</sup> can replace the table, depending on how you want to realize the data. In the table, Exe refers to the execution time in either seconds, microseconds, nanoseconds, or machine cycles. You can choose the parameter, but you have to be consistent. Ops refers to the total number of operations needed to effect the procedure element.

The size and activity of the model need to be spread apart. For example, the table shows such operational characteristics as operands and labels. When you set up the characteristics as defined by the table, you define the overall system architecture.

In establishing the uniformity of the model, you take into account that each compiler has unique ratios between

source-code operations (and operands) and the number of processor instructions. For optimized compilations this number ranges from 1.8 to 2.4. Any high-level verb takes at least 1.8 instructions.

Using these numbers as a basis, I've constructed the notion of a mean term, or an MOP (mean operating profile), to more accurately picture compiler-to-instruction-set interaction.

### Assumptions

In any model you have to make some assumptions. I made the following. The system uses a 20-MHz 386. The system latency equals *c* (the speed of light, which can be ignored). We are operating in a domain so slow that any latency caused by bus transfers is minimal. Each instruction takes 10 machine cycles. The SDP (standard data processor) Ada code is highly optimized and uses firmware and OS calls. Only some portions of any environment are resident in memory at any one time, based on OS scheduling and the definition of the processing environment.

So that the environment can be properly represented, I've modified Huck and Flynn's approach to determining the extent of the environment:

$$(\log_2 \text{MOP}) \times A_1 + (\log_2 A_1) \times B_1 (\log_2 B_2) \times C_1$$

in which *A*, *B*, and *C* represent the subordinate procedures that make up the environment of a CSCI. This for-

**System profile based on CI measurements.**

Procedure	Exe	Ops	Operands	Labels	Branches	
					Taken	Entered
Test for A	100	—	—	—	—	33
If stat = A	100	1	3	—	20	—
Do no test	100	1	3	1	—	—
Else test	100	5	2	1	1	—



mula depicts the total size of the environment. Note that the MOP differs for each environment. In one processor, some functions can represent several environments. A complex relationship exists between each environment and the state of the environment(s) at any given point in time.

The dynamic instruction count is a primary measure of the architecture. This count tells us how efficiently the system executes a program (no matter how badly the program is written). Because we have no page faults to contend with, the context remains constant (contrary to Motorola's processors). The function is a copy-in, copy-out process.

The static program size has a secondary effect on the architecture and optimization. However, this size represents the impact on system memory. This designation doesn't imply that a 2-Mbyte program uses 2 Mbytes of memory at one time. Rather, it establishes the upper boundary.

Since much of the work performed in any given processor/memory system involves fetching data objects, heavy processing tasks require only moderate concern. Exceptions do exist, of course. The notion is to preserve as much processing resource (memory and I/O bandwidth) as possible.

## Optimum values

To define the optimum values of instructions to executable functions, you must determine the Gibson Mix.<sup>4</sup> You do this by analyzing the efficiency of the compiler and the overall software architecture. The OS can perturb the Gibson Mix by forcing double execution of certain instruction types. For example, improper interaction can cause multiple memory-to-memory moves.

You can use such tools as  $J$ ,  $K$ , and CPU utilization to model the system for perspective on the type of user operation you seek:

$$\text{Elapsed time} = (J + K) (\text{CPU intensity})$$

$$\text{Page rate} = 2 (\text{number of pages}) / J$$

Elapsed time is measured in seconds per Mbyte. You must define the page (a 386 typically has a 4-Kbyte page). You can, if desired, tabulate this data for later use to create a graphics representation. I recommend that you use a convenient tool such as an Excel spreadsheet to enter the data and perform the calculations.

Now you have several ways to analyze the modeling process. The next task is to plot the data from this table: elapsed time versus CPU intensity and CPU utilization versus CPU intensity.

You can consider one final model that measures the CPU utilization factor. Kenniston W. Lord, Jr., suggested the following method in the *CDP Review Manual, A Data Processing Handbook*.<sup>5</sup>

$$\text{CPU}_{\text{idle}} = EC_1 * \text{scan}_{\text{idle}} * I * 1,000 / \text{interval}$$

$$\text{CPU}_{\text{wait}} = EC_2 * \text{scan}_{\text{idle}} + EC_3 / EC_2 * \text{scan}_{\text{wait}} * I * 1,000 / \text{interval}$$

$$\text{CPU}_{\text{util}} = 100 - (\text{CPU}_{\text{idle}} + \text{CPU}_{\text{wait}})$$

Here,  $EC_1$  is the number of times no dispatches exist (idle count),  $EC_2$  is the busy count (a lot of dispatches), and  $EC_3$  is the task waiting for an event.  $\text{Scan}_{\text{idle}}$  represents the number of instructions executed at idle,  $\text{scan}_{\text{wait}}$  is the number of instructions executed to determine if a task is waiting, and  $I$  equals the instruction-execution time in  $\text{scan}_{\text{idle}}$ . This time can have many values. For our work, we assume average instruction time.

## The bottom line

The bottom line is that you can model the system. You should come out with a utilization-factor graph that will give you an idea of how best to use a processor/memory system.

The overall notion here is to pro-

vide a foundation for a utilization standard. Much of this work is still in development. The Huck and Flynn book forms a solid foundation. If you have an interest in this type of performance measurement, please drop me a note.

## References

1. J.C. Huck and M.J. Flynn, *Analyzing Computer Architectures*, IEEE Computer Society Press, Los Alamitos, Calif., 1989.
2. J.V. Jones, *Engineering Design Reliability, Maintainability, and Testability*, TAB Books, Inc., Blue Ridge Summit, Penn., 1988.
3. W.H. Press, et al., *Numerical Recipes: The Art of Scientific Computing*, Cambridge Academic Press, Cambridge, England, 1986.
4. J.C. Gibson, "The Gibson Mix," Tech. Report TR00-2043, IBM, Poughkeepsie, N.Y., 1970.
5. K.W. Lord, Jr., *CDP Review Manual, A Data Processing Handbook*, Fourth ed.; Van Nostrand Reinhold Co., N.Y., 1986.

## Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 189    Medium 190    High 191

## Micro Review

Richard Mateosian

2919 Forest Avenue

Berkeley, CA

94705-1310

(415) 540-7745

### Desk planners plus

***The 1991 Computer Desk Reference and Appointment Calendar*** (Microsoft Press, Redmond, Wash., 224 pp., \$19.95)

Every year I structure my life around some sort of appointment book. If you do this too, you may find this calendar interesting.

The book measures approximately 8 inches by 10 inches and provides a two-page spread for each week of the year. Monday through Friday appear in columns of their own. Each column holds lines labeled 7 through 6 plus LUNCH and EVE. Then there are sections for KEY TASKS, PHONE OR WRITE, and EXPENSES. Along the bottom of the two-page spread are calendars for 12 months, centered on the current month. Saturday and Sunday and the quotation of the week share a column.

Many of the quotations are specific to the computer field. One of my favorites is "Advertising cannot overcome major marketing flaws in your program."—Andrew Donchak. Another is "Good design keeps the user happy, the manufacturer in the black and the aesthete unoffended."—Raymond Loewy.

The book provides the usual places for personal data and for names and addresses. It also provides a quarterly travel planner. Each calendar quarter uses one page, and many computer industry trade shows are shaded in. The book also allows two-page spreads for each of 1991 and 1992. In these, it allocates one small rectangle for each day.

The reference material included in the book is quite useful. I especially like the minitravel guides provided for Atlanta, Boston, Chicago, Dallas, Denver, Houston, Los Angeles, New

Orleans, New York, San Francisco, Seattle, and Washington. Each is a two-page spread with an area map on one page. The other page contains a downtown map, some general information, and a few hotels and restaurants. I have no idea how Microsoft selected the hotels and restaurants.

The book contains many other reference pages related to travel. It also contains a two-page summary of Dataquest statistics and many pages of lists of companies, organizations, and trade shows. It also offers a feeble technical reference section, which contains Intel instruction sets and industry-standard character sets.

I might not have designed this book exactly the way Microsoft did, but I think it is well designed, and I think many people in the computer industry will find it useful. And if you're looking for a present for the computer professional who has everything, this might just fill the bill.

***Cache and Memory Hierarchy Design—A Performance-Directed Approach***, Steven A. Przybylski (Morgan Kaufmann, San Mateo, Calif., 1990, 223 pp., \$33.95)

This is a reworking of the author's 1988 thesis, which he wrote at Stanford under the tutelage of John Hennessy and Mark Horowitz. Not surprisingly, Przybylski performs the same kind of quantitative analysis that Hennessy and Patterson advocate in their recent book on computer architecture (see Micro Review, June 1990).



Przybylski measures cache design with the metric of program execution time. He contrasts this with the time-independent metric of cache miss rate. Execution time is more useful than miss rate but requires an elaborate analysis, which Przybylski provides.

Przybylski views cache design from a system perspective. He recognizes that when CPU and cache are designed together, a more nearly optimal solution can emerge. Of course, the user of a commercial microprocessor must take the CPU as given and design a cache around it, but Przybylski's analysis still applies in the most interesting cases.

Przybylski first analyzes the problem for a single-level cache, then shows how to determine the optimal multilevel memory hierarchy. Of course, he does not present a formula or even an algorithm to determine the optimum cache parameters for a given system. However, he does show how to evaluate the various possible tradeoffs quantitatively.

Przybylski complements his analysis with simulations. He uses a sophisticated simulator and a variety of program traces to generate numerical results of "experiments."

Computer system designers and computer science students will find this book interesting. The book is not for the casual reader, but Przybylski provides separate summaries of findings for readers who wish to avoid the gory details.

***The Elements of Spreadsheet Style***, John M. Nevison (Simon & Schuster, New York, 1987, 212 pp., \$12.95.)

I used to worry that we were moving backward. Just as structured programming had begun to rationalize mainframe programming, the personal computer created large hordes of Basic programmers and spreadsheet writers. These mass programming media lacked the tools for systematic programming. Practitioners of the new arts

reinvented spaghetti code, data entangled with programs, comment-free listings, and trial-and-error design. Modularity went out the window. What a mess!

Things have progressed. Basic has given way to C—an improvement, but not a solution—and now John Nevison has stepped forth to try to enlighten the spreadsheet hacker.

Like *The Elements of Programming Style* by Kernighan and Plaugher, a classic from the early 1970s, Nevison's book was inspired by *The Elements of Style* by Strunk and White. As an admirer of both of those works, I began to read Nevison's book with interest and enthusiasm.

Nevison sets forth 22 rules of spreadsheet design. He devotes most of the book to justifying and illustrating these rules. He also talks a little about reusing spreadsheets and building collections of spreadsheets. I don't think that most spreadsheet programs contain adequate tools to support a truly modular approach, but I'm glad that someone is addressing the issue.

Nevison's rules are

1. Make a formal introduction
2. Title to tell
3. Declare the model's purpose
4. Give clear instructions
5. Reference critical ideas
6. Map the contents
7. Identify the data
8. Surface and label every assumption
9. Model to explain
10. Point to the right source
11. First design on paper
12. Test and edit
13. Keep it visible
14. Space so the spreadsheet can be easily read
15. Give a new function a new area
16. Report to your reader
17. Graph to illuminate
18. Import with care

19. Verify critical work
20. Control all macros
21. Focus the model's activity
22. Enter carefully.

Some of these rules don't mean what you might think, so you'll have to read the book to get full benefit from the list. For example, the 22nd rule has nothing to do with typing errors. Nevison is telling you to control the entry of data into submodels, just as you would declare subroutine parameters in a C program.

Nevison illustrates these rules with actual spreadsheets. This could be incredibly boring, but he avoids that problem with humor. He bases his spreadsheets on nursery rhyme themes and embeds little historical puzzles in them.

If you write a lot of spreadsheets, or if you don't because they don't seem to apply to your problems, take a look at this book. It's worth some study.

---

### Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 177    Medium 178    High 179

---

# Software Report



David K. Kahaner

US Office of Naval

Research, Far East

kahaner@xroads.cc.

u-tokyo.ac.jp

## Quality improvement

**S**ome Japanese companies manage the software development process better than comparable US companies. In addition, numerical and nonnumerical software developers need more interaction.

These major conclusions surfaced at last winter's Second International Workshop on Software Quality Improvement in Kyoto, Japan. I summarize these meetings from the perspective of a numerical analyst.

### Goals

Sponsored jointly by the Ministry of International Trade and Industry (MITI), the Joint System Development Corporation (JSD), and the Research Institute for Software Engineering (RISE), the workshop brought together almost 60 scientists. About a dozen came from the United States, one from Italy, and the remainder from Japan. Koji Torii of Osaka University and Victor Basili from the University of Maryland jointly organized the workshop.

The official goal of the Software Quality workshop was to promote the use of quantitative and qualitative evaluation. This use will help us understand the effects of various software processes and how to improve software measurement techniques. In 1990 billions of dollars were spent on computer software; the US Navy alone spent \$10 billion. In the future this figure will rise. Logically therefore, we must conduct research in methods that can lead to reduced rates of increase in these costs. All large technology organizations face a similar challenge.

### KRP

The workshop was held at the Kyoto Research Park, a new and spectacularly modern facility about 15 minutes by taxi from the Kyoto train

station. KRP bills itself as a "self-contained laboratory complex with all the benefits of a modern city." Apparently with help from the city of Kyoto, KRP continues to construct and manage this large facility, which commenced operation in the fall of 1989.

The sessions offered simultaneous translation in Japanese and English so participants could use the most natural language. (The wireless receivers were German.) Last year's English-only workshop proved to be a deterrent to many of the Japanese participants.

### JSD

About 100 related businesses and 19 major information processing companies founded JSD in 1976. JSD seeks to improve technical standards and to strengthen the Japanese information processing industry. JSD focuses on national projects, development of a paperless system for the Japanese Patent Office, distribution of the Spider image-processing package, cooperation with the Small Business Promotion Corporation, and software sales promotions.

### Observations

As a scientist who has been developing software throughout my career, I realized after listening for two days that the issues addressed here apply to a very different class of problems than those faced by myself and my peers. The best sense of this came to me while listening to the speaker from Mitsubishi. M. Masuda claimed that his company produces more than 34 million lines of software (code) each year. Most of this is developed to run industrial processes such as manufacturing production lines.

The main questions associated with this software are not whether the algorithm is clever,



efficient, or creative. They concern how to reduce the number of potential errors that large, multiperson software projects can produce. In fact the most important parameter discussed at this workshop was BKLOC, or bugs (errors) per thousand lines of code. This community appears to have little room for the traditional questions about numerical methods that physicists, chemists, and engineers raise when thinking about computer programs.

I concluded first that a general perception existed that the Japanese (companies) were doing a better job than their US counterparts in reducing error rates. Basili claimed that Japanese BKLOC rates were about one order-of-magnitude less than those in the US, down below 1 BKLOC. Further, he said that Japanese software productivity was from two to 20 times that in the US. He claimed the major reasons were the Japanese companies' better management techniques and better methods of motivating people. (After listening to several talks by representatives of Japanese industry, I concurred with this assessment.)

Nevertheless, the gap between industry and research was greater in Japan than in the US. Basili felt entry-level Japanese software developers are less well prepared than their US counterparts. But it reverses after about five years of work. Japanese companies do a good job of training and retraining workers, so perhaps this accounts for part of the turnaround effect. (On a personal level I have been amazed to discover courses in Unix, mathematics beyond calculus, and statistics on the educational channel of our home TV during prime time.) Cohost Torii echoed the remarks about the narrow pipeline between Japanese university research and industry.

Secondly, I was surprised by the meager attendance and intellectual presence of US industry, even considering the distance to Kyoto. W. Agresti from Mitre Corporation discussed the ways to assure quality in

software being developed by others. M. Deutsch (Hughes Aircraft) discussed an after-the-fact analysis he had done of some of Hughes' projects while he was on sabbatical. F. McGarry (NASA) discussed work done to a large extent in collaboration with Basili at Maryland. Representatives of CTA Inc., a small Maryland contractor, and MCC, a Texas software think tank, also presented papers.

---

***Pictographic kanji  
lets readers grasp  
the essence of a  
page more  
quickly.***

---

Some other American companies attended but presented no papers. Representatives from AT&T Bell Labs (Columbus, Ohio, office only) and Gen Rad in Massachusetts wanted to learn more about Quality Function Deployment, a method developed by Tadashi Yoshizawa of Tsukuba University. QFD is well known and has been promoted by Yoshizawa since the 1970s. M. Ohba from IBM Japan spoke only about modeling the errors that were discovered after software was delivered. (I questioned his use of linear differential equations as being simplistic.)

On the other hand, Japanese industry was well represented. Oki Electric, NEC, Mitsubishi, Fujitsu, Nippon Steel, Sharp, and JSD papers gave very clear evidence that these companies try hard to learn how to improve software productivity. Papers by university researchers from both countries were mostly excellent.

I was struck by the fact that the Japanese really work at reducing software errors. They strongly emphasize user requirements driving the

design process (natural in any company that produces a product), a team approach to solving problems, developing methods that assure quality at every step of the software process, and quality control involving all departments including top management. The speaker from Nippon Steel said it very well, "We put stress on human motivation as well as machine's automation."

### **Selected details**

The Tetsuo Tamai (Tsukuba University) paper on Japanese-based programming tools generated a substantial amount of discussion. Essentially, Tamai studied the existing programming languages that use Japanese in some direct way and attempted to compare them. The discussion centered not so much on his results, which were very tentative and modest, but on the general usefulness of a Japanese programming language. Basili pointed out earlier that he had been impressed with the strides made in the use of Japanese syllabic input (hiragana) with resulting conversion to kanji. I have watched this process, too. Although it is slower than using an ordinary alphabet, it is only a factor-of-two slower. It allows Japanese speakers more natural access to computers.

One of my noncomputer colleagues remarked that pictographic kanji permits experienced readers to grasp the essence of a page more quickly than alphabetic languages. Other icon-based software (Sun, Macintosh, and so on) are also pictographic and clearly very successful. Jun Murai (University of Tokyo) wrote recently that Japanese network traffic is almost all in kanji. Perhaps Japanese prefer to communicate this way. In any case it was admitted that Japanese language programming would be unmarketable outside the country. Consequently research in this direction has never been a priority project.

I believe Japanese-language software would provide improved productiv-

ity. It would open the door to amateur software developers in much the same way that the PC did in the US. Further, the lessons learned from developing Japanese software could then be applied to other Asian languages for countries whose markets are just now opening for computer technology. Even the current wave of Japanese word processors has already fueled very good display technology. A Japanese PC on a secretary's desk has much better graphics capability than the US counterpart.

The paper on JSD's Faset project interested me because it represented one of the major projects undertaken by this group. I was disappointed with the progress they have made. Some other listeners seemed to feel similarly. They questioned the use of inefficient Lisp as an output language. They wondered why logic programming (Prolog) was not used, and why the tools were not better integrated. This work strikes me as being so high level that it suffers from a lack of concrete focus.

John Knight (University of Virginia) is a recognized expert on safety critical software, such as those used in nuclear power facilities, avionics systems, or dangerous medical appliances. He discussed the problems associated with designing software with failure rates of  $10^{-9}$  per event or per hour. For extremely reliable hardware systems the usual approach is redundancy—shown both by analysis and experiment to be effective. For software the same technique is often used. That is, different contractors develop software with identical specifications. Unfortunately, Knight's research shows that the usual assumption of independence between failures of redundant systems is often invalid for software. Difficult parts of the design are likely to lead to errors no matter who codes them. He concluded very succinctly, that he was "scared to death" when he thinks about some of the places such safety-critical software are used.

McGarry's paper was particularly

relevant to me. He provided comparisons between software projects using Fortran and Ada. He showed that, surprisingly, the percentage of time spent in the design, test, and code stages of a big project was about the same with either language. Further, the error profiles were also about the same, although Ada programs had fewer interface errors. The US NASA approach to improving software quality lies in "understanding" many of the measures of software quality that have been developed over the past 15 years, "analysis" or defining relationships between the software process and software product, and "automation" or the development of software tools to improve quality. Much of the research has been done within the Software Engineering Laboratory jointly supported by NASA, the University of Maryland, and Computer Sciences Corp.

T. Yamazaki represented Nippon Steel's view that employee motivation and technology are like two wheels on the same axle. Both must have equal diameters, or straight-line motion toward the company's goal cannot occur.

A similar view was expressed by Masanori Teramoto. He estimated that NEC employs approximately 15,000 people in 2,500 groups associated with software development. They hold meetings in production design, software quality control, and evaluation and self help. The meetings involve all levels of company employees, including top management, and are very goal oriented.

NEC estimates that in 1991 it will write 140 million lines of code. It expects BKLOC to be less than 0.1. Further, its reuse percentage (code that can be used in other applications without rewriting) will increase from 50 percent in 1987 to 60 percent in 1991.

As a side remark, Teramoto noted that Unix helped increase NEC productivity. The same system is used across a wide variety of computers, and

hence less relearning was needed. My own experience is that Unix is a wonderful system for software hackers who love its flexibility and abbreviated command syntax. It is also an awful system for the inexperienced who are easily confused by these same features.

## Miscellaneous

An article in *Japan Times* by John Boyd, entitled "Is a hard fall awaiting American soft?" presents some conclusions that dovetail with the remarks about Japanese word processors. Boyd details a speech made by Bill Totten, president of Ashisuto K.K., a Japanese importer and distributor of foreign software. Some excerpts follow:

The worst mistake (Americans) could make was to underestimate the Japanese software competition. I think the American software industry needs to face up to the fact that things are changing rapidly. And unless they make substantial changes in the way they're handling their business, they're going to lose this industry like they lost the television, automobile, and a lot of other industries. The reasons behind the change sound frighteningly familiar: high US costs and prices compared to Japanese offerings; a parochial business focus and an unwillingness on the part of many US software companies to even listen to the needs of overseas customers.

The 2-byte character code is a good example. Virtually all Japanese software uses these 2-byte codes, which are necessary to represent the 3,000 or more kanji characters. Several years ago IBM issued a new standard to support the 2-byte code, but few US software companies currently use it because it isn't necessary in the US or Europe. Totten says,



Japanese companies are coming out with a lot of software that is now functionally the same as American software, but is better manufactured, [has] few bugs, more clearly documented source code, and is smaller and faster...US companies are on the verge of losing the Japanese market, and as Japanese distributors grow and try to expand to Asia and Europe these markets will be lost too.

My own experience so far is that these observations do not yet apply to scientific software. I have seen very few examples of Japanese engineering software in use. In fact, I have repeatedly been told that the main reason that Japanese scientists like the Cray rather than NEC, Fujitsu, or Hitachi supercomputers is not because Cray computers are faster. It is because so many well-tuned engineering and analysis packages run on Crays. Unless Japanese supercomputers sell well, software vendors will not be very motivated to create versions that are specific to them.

In the PC and workstation world things are quite different. Any computer with a graphical or menu interface is likely to run Japanese software because of the need to represent kanji.

*[Editorial Board member David Kabaner travels in Japan on assignment with the US Office of Naval Research, Far East. His comments are his own; they do not express any official policy.]*

### Reader Interest Survey

Indicate your interest in this department by circling the appropriate number of the Reader Service Card.

Low 215 Medium 216 High 217



### Optics: The next wave?

Positioning itself for an anticipated "next wave" in the electronics industry—electrooptics and integrated optics—is Atlanta's Georgia Institute of Technology. Researchers there believe a recently completed 100,000-sq ft addition to the Microelectronics Research Center will contribute to Tech's attempts to understand optic device technology.

The three-story Center building contains a "photon corridor" for optical research (in which researchers pipe laser beams), 7,000 sq ft of clean room space, a laboratory, and office and

conference room space for 40 researchers and 80 students. In the photon corridor, the Center's optics laboratories share more than a million dollars' worth of laser equipment.

Center researchers—in conjunction with industry partnerships—study the devices, which communicate to and from optical fibers and are too costly and risky for one company to develop. They investigate ways of using electron wave effects in efforts to build 1-billion-bps circuits. Much of the research focuses on developing high-efficiency solar cells that combine silicon with other materials and analog silicon

### Micro bits

Merit, IBM, and MCI established Advanced Network and Service, Inc., a not-for-profit organization to manage and operate the federally funded **NSFNET**.

Concerned about **trade with the European Community?** US interested parties may now telephone (301) 921-4164 for a recorded hotline message on draft EC laws and standards that might create technical trade barriers.

Open Software Foundation announces release of its Unix-based

operating system, **OSF/1**. OSF/1 features built-in multiprocessing, a microkernel, and added security.

On April 8 the US Federal Communications Commission begins testing six US, European, and Japanese proposals in efforts to select a **US HDTV standard** by 1993.

The US National Research Council recently created the **Scientific and Technical Information Board** to find ways to structure, manage, and communicate scientific and research data sets.

devices based on neural networks for applications in instrumentation.

Close ties with Tech's Manufacturing Research Center offer the capability for developing the technology and moving it into production. This chance to work together with exotic materials or combine electronics with photonics allows researchers to maintain control of the complete process and permits fine-tuning as needed. "We can fabricate new ICs," says the Research Center's director, Richard Higgins, "and they [manufacturing] can determine how to package and assemble them into reliable working equipment."

### RS/6000 leads in performance

Testing of recent RISC products by Workstation Laboratories places the IBM RS/6000 performance ratings above those of other well-known RISCs. The 6000 achieved a rating of 13,594 Khornerstones in six floating-point-intensive tests written in C and Fortran. The DECstation 5000 came in at 10,417 and the Mips Magnum (RS3230) at 10,567 Khornerstones.

As the table below shows, the 6000 also led in vector and transaction processing tests. Vector tests require completion of 100,000,000 multiplication operations using multiple  $25 \times 25$  matrices. The transaction processing test represents multiuser performance and is highly disk-intensive.

### Learn about LANs on your own time

A recently released self-study telecommunications series can help you learn how to plan and design a tailored local area network system. You can learn how to implement a system that is already on site or integrate systems that may have been inherited through a merger or acquisition.

The six-course series from Science Research Associates details the IEEE 802.2/3/4/5 LAN protocols. The 32210 courses adhere to the models and conventions of the IEEE and the standard OSI model. Available individually or as a series, the courses can be licensed for 60 days or one year from SRA, 155 North Wacker Drive, Chicago, IL 60606-1780.

Ask for the *Local Area Network Architectures and Implementation Series*.

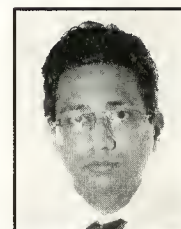
### Multichip modules

Application-specific electronic subassemblies called multichip modules may soon find their way into electronic equipment applications. MCMs provide space-saving interconnection for bare or unpackaged semiconductor chips, which are then protected by a coating or an enclosure.

A Business Communications Company, Inc. study predicts that MCMs will become an important ingredient for high-performance electronic systems

incorporating CMOS and VLSI technologies, as well as high-speed systems. BCC predicts that this market will total \$200 million in 1990 and \$896 million by 1995. Joseph Castrovilla, BCC electronics analyst and study author, states that these "subassemblies will provide solutions to meet the requirements for advanced electronic equipment for state-of-the-art component density, high speed, and reduced size."

Contact BCC, 25 Van Zant Street, Norwalk, CT 06855, for a copy of the \$2,650 *Multichip Modules in Electronics: New Technologies and Markets*.



### Hootman adds to Board

Editor-in-Chief Joe Hootman recently appointed Ashis Khan, to the Editorial Board of *IEEE Micro* to review submitted manuscripts. As a senior technical specialist at Mips Computer Systems, Khan consults with customers designing systems based on the company's RISC architecture. He has written articles and delivered public seminars on that topic.

Khan earned his MSEE degree at State University of New York at Stony Brook and his BTech degree at Indian Institute of Technology in Kharagpur. He is a senior member of the IEEE.

Benchmark data.

System	CPU/MHz	Floating-point Khornerstones	Vector	Transaction processing
IBM RS/6000/320	IBM RISC/20	13,594	5,882,352	35.64
DECstation 5000	R3000/25	10,417	1,677,852	18.29
200 cx				
Mips RS3230/ Magnum	R3000/25	10,567	2,207,505	11.96
Sun Sparcstation I	Sparc/20	3,778	628,141	19.15
Tektronix XD88/10	88000/20	5,645	645,161	16.42
Compaq DP486/25	i486/25	6,311	544,366	18.71

### Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 183    Medium 184    High 185





## New Products

Send announcements of new microcomputer and microprocessor products to  
Managing Editor, IEEE Micro; PO Box 3014, Los Alamitos, CA 90720-1264.

Joe Hootman

University of

North Dakota

### DSP and related products

#### ■ Code development enhanced

DSPlay XL 3.16 code generation software expands available DSP and related functions by more than one third over its previous version. It includes a graphic editor, function library, graphic displays, filter design programs, assembler, and real-time analog I/O support. Users create code using block diagrams from the standard library or customize their own using the built-in assembler. The provided monitor offers debugging features. DSPlay XL 3.16 supports AT&T WE DSP32 and DSP32C processors. *Burr-Brown Corporation; \$1,495, available from stock.*

**Reader Service No. 10**

#### ■ Software analyzes ADC performance

ZPA1000 software evaluates the performance of analog-to-digital converters (ADCs) by analyzing signals in the time and frequency domains, and producing linearity measurements. The mouse-driven software operates as a digitizing oscilloscope, spectrum analyzer, and histogram analyzer, accepting analog inputs from 16 bits at 150 kHz to 12 bits, 10 MHz. The save/load function evaluates converter performance and system parameters at the final test stage. ZPA1000 allows for hard-copy printouts on designated graphics or laser printers. *Burr-Brown Corporation; \$995 in unit quantities from stock.*

**Reader Service No. 11**

#### ■ C compiler supports signal processor

An ANSI C optimizing compiler for Texas Instruments' TMS320C25 digital signal processor generates assembly language, supports the writing of C-level interrupts, and supplies C libraries in source form. The compiler, which supports the TI assembly format, uses the DSP

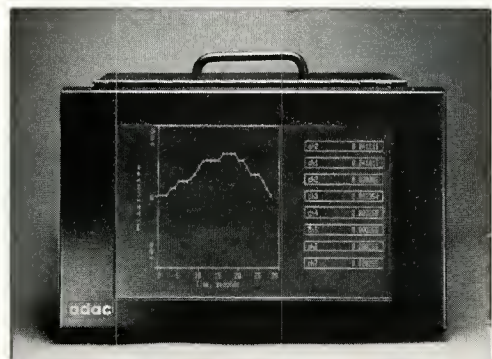
chip to create optimized code. Its companion assembler produces relocatable object code, listing files, and diagnostic messages. Assembler utilities include librarian, cross-reference generator, and object code converters. The assembler supports Hewlett Packard- and TI-tagged formats, and accepts TI assembly mnemonics and directives. *BSO/Tasking; from \$1,695 for the compiler, assembler, linker, and librarian.*

**Reader Service No. 12**

#### ■ Midas touch for industrial data

Measuring 7 inches wide, 10 inches deep, and 6 inches high, the Midas-150 PC provides industrial data acquisition and process control. Its features include a 16-MHz 286-compatible processor, 1 Mbyte of RAM, three storage options (1.44 Mbyte, 3.5-inch floppy; 40-Mbyte hard drive; or semiconductor RAM disk), a seven-slot passive backplane, and integral field wiring screw terminations. The Midas-150 uses compatible process control software, such as Genesis, LT/Control, and Lab Tech Notebook. *ADAC Corporation; from \$3,200.*

**Reader Service No. 13**



*Midas-150 industrial PC*

### ■ A/D board offers 100-kHz transfers

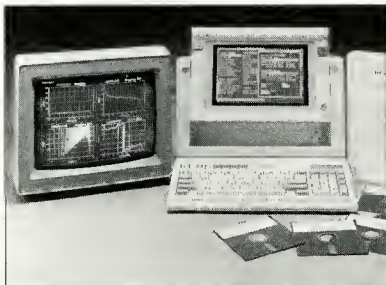
The DT2812 analog and digital I/O board conducts direct memory access (DMA) transfers at rates up to 60 kHz. Two digital-analog converters (DAC), three 16-bit counter/timers, program-mable pacer clock, and interrupt supports allow users to work in scientific and industrial applications. The DT2812-A board offers maximum digital-to-analog and analog-to-digital transfer rates of 100 kHz. Both boards are IBM compatible. *Data Translation*; \$629 OEM (DT2812), \$695 OEM (DT2812-A).

**Reader Service No. 14**

### ■ DSP program features graphics

A DSP analysis software package with accompanying graphics capability performs calculations and analysis using one-dimensional fast Fourier transforms. The program—called Fourier Perspective III—features digital and median window filtering, linear systems functions, windowing, convolution and correlations, and a number of mathematic functions for conducting analysis. Fourier's Data View saves graphics in encapsulated Postscript and other formats, and it outputs to a range of printers and plotters. *Alligator Technologies*; \$695, available one week ARO.

**Reader Service No. 15**



*Fourier Perspective III software system*

### ■ Triggers set aim for analysis

The triggering system of the R380 spectrum analyzer and digital oscillo-

scope permits precise setting levels using on-screen tools or remote control. Connected to a serial port, the R380 hardware includes two 14-bit A/D channels; input signal gain ranges; and a signal cursor and two markers to measure voltage, time difference, and frequency. *Rapid Systems*; \$1,995.

**Reader Service No. 16**

### ■ Voice processing unit for XT's

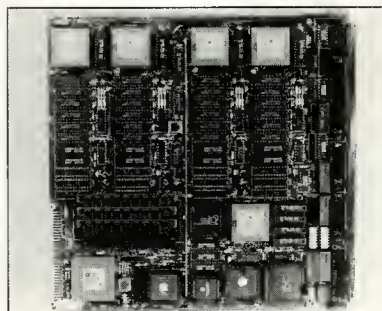
A four-port voice processing component allows developers to construct DSP-based systems on 8088 and 8086 platforms. The new IBM XT-compatible RDSP/4108 uses an 8-bit hardware interface to the host computer. The RDSP/4108 can work with the 16-bit RDSP/4208 voice processing unit in an 80286 or 80386 host computer. *Rhetorex*; \$995.

**Reader Service No. 17**

### ■ DSP board available for Next

A DSP-based board for the Next computer features five 27-MHz digital signal processors to execute audio and communications applications. Four of the DSP56001 chips on this Quint Processor board function as slave processors to perform computation; the fifth operates as a I/O processor that manages dynamic RAM, SCSI mass storage, and interprocessor communications. The Quint Processor contains five DSP ports for analog and digital I/O. *Ariel Corporation*; \$6,995, available in 30 days.

**Reader Service No. 18**



*Quint Processor DSP board*

### ■ Coprocessor cards support Suns

Two coprocessor cards operating at 13.5 MIPS (million instructions per second) conduct DSP and I/O for Sun Microsystems' Sparcstation and S-bus-compatible computers. The S-56 and S-56X cards, bundled with debugging and library tools, process real-time tasks in Unix-based environments. They also include up to 192 Kbytes of zero-wait-state memory and a Next-compatible DSP port. *Ariel Corporation*; from \$2,495 or \$1,495 OEM (S-56), \$2,995 or \$1,995 OEM (S-56X).

**Reader Service No. 19**

### ■ Self-calibrating converter adjusts for errors

A 12-bit, plus-sign, analog-to-digital converter calibrates itself to adjust linearity and zero errors. According to the company, the ADC1251 corrects internal errors, ensuring no missing codes over temperature with zero errors of  $\pm 1$  LSB (last significant bit) and full-scale errors of  $\pm 1.5$  LSB. Available in a 24-pin, dual-in-line package, the ADC1251 accomplishes conversions in 8  $\mu$ s. A sample-and-hold function allows its use in DSP applications. *National Semiconductor*; \$17.50 (ADC1251CJ, 100s).

**Reader Service No. 20**

### ■ Driving A/D channels simultaneously

The TSI 98304 coprocessor drives A/D and D/A converter channels simultaneously up to 20K samples/second per channel. These eight 8-bit channels work with a TMS320C25 digital signal processor with up to 2 Mbytes of dual-ported data memory. The memory provides buffering between the real-time I/O system and the time-shared processes for an uninterrupted flow of data. The TSI 98304 fits the DIO-II slots in the HP 9000 series 300 and 400 workstations. *Tetra Systems*; \$3,995 (2 Mbytes) or \$2,995 (no dual-ported memory), delivery 6-8 weeks ARO.

**Reader Service No. 21**



### ■ Array processor performs at 33 Mflops

Compatible with VMEbus systems, the DSP200 general-purpose, floating-point array processor performs at a maximum of 33 Mflops using a Texas Instruments CPU. Its P2 connector implements a Quad 16-bit I/O bus for processor data interchange. This bus conducts data transfers at 8 MHz in one clock cycle. Twelve-bit converters support analog I/O, and front-panel connectors support 64 bits of digital I/O. *Olsson Engineering; \$8,495, delivery 60 days ARO.*

**Reader Service No. 22**

### ■ Converter/amplifier digitizes input signals

A T/H amplifier combines with a sampling A/D converter to digitize static and dynamic analog input signals. The MN6774, based on the MN774 12-bit, 8  $\mu$ s A/D converter, targets military, aerospace, and industrial applications. J, K, S, and T models address differing levels of performance, including unipolar error, zero error, and absolute accuracy specifications. *Micro Networks; from \$105 (100s); delivery in 8-12 weeks.*

**Reader Service No. 23**

### ■ Card offers 12-bit resolution

A full-size data acquisition board offers 12-bit resolution for both A/D and D/A conversions. The R812G includes software-selectable A/D input ranges, built-in interrupt and direct memory access, two D/A outputs, and programmable gains of 1, 2, 4, 8, and 16. The card can execute the following applications: temperature, pressure, proximity, level, height, distance, and force. *Rapid Systems; \$595.*

**Reader Service No. 24**

### ■ Interface reduces noise

Designed for applications requiring exact analog voltages, the DA700 interface features six or eight channels of 12-bit conversion with closely matched digital/analog converters and

output amplifiers. Each analog signal couples with a ground signal, reducing noise pickup and increasing accuracy at output. Developed for the PC/XT/AT bus, DA700 supports jumper-selectable unipolar or bipolar operation. *Real Time Devices; \$297 (six channel) or \$359 (eight channel), stock to two weeks ARO.*

**Reader Service No. 25**

## Design tools

### ■ Analysis software optimizes circuit design

The Clio software package lets designers create a match between design and desired performance with automatic circuit optimization. Clio also features yield analysis, design centering, schematic capture, design specification entry, circuit simulation and analysis, and results processing and presentation. The user can control the design, analysis, and design processes through component menus and mouse operations on Sun Microsystems and Hewlett Packard/Apollo platforms. *Electrical Engineering Software; \$18,750 (network configuration of four seats).*

**Reader Service No. 26**

### ■ Analog circuit program increases plot output

The 2.50 version of ECA-2 software upgrade expands the interactive output of multiple, worst-case, and Monte Carlo plots. It provides linear and/or dB values, phase and/or phase delay, and component sweeping to optimize the circuit or predict end of life. A built-in editor, real-time graphics, and fundamental analog simulation capabilities are other features. ECA-2 2.50 requires a minimum of 256K of RAM and MS-DOS 2.0 or later to run on IBM-compatible computers. *Tatum Labs; \$775.*

**Reader Service No. 27**

### ■ Spicing up circuit simulation

A software program simulates circuits down to path delays critical for 40-MHz-or-greater digital clock fre-

quencies in designs for PCBs or ICs. Contecspice, based on the public domain Spice3C1 circuit simulator, provides algorithms for modeling coupled lossy transmission lines. This mixed-level, mixed-mode program targets engineers who design high-performance PCBs, digital and analog ICs, and device packaging. *Contec Microelectronics USA; \$1,998 (PC) or \$5,419 (Sun workstation).*

**Reader Service No. 28**

### ■ Schematics generated and checked

The Vutrax-II GES schematic entry program includes features for checking drawing accuracy and analyzing critical unconnected inputs. Users can generate circuit diagrams in many text fonts and line widths, and subcircuits and repetitive figures may be boxed and duplicated. Vutrax-II interfaces with analog and digital simulation programs from Tatum Labs and other companies. It requires a hard disk with 4 Mbytes available. *Tatum Labs; \$495.*

**Reader Service No. 29**

### ■ Design tools for ASIC

A design kit for application-specific ICs offers a range of tools for the Dazix design environment on the Sun-4 workstations. Cell symbols, simulation models, net list and test pattern converters, and an ASIC management environment come in each Fujitsu kit. The available libraries cover technologies such as complementary metal-oxide semiconductor (CMOS), bipolar CMOS, and emitter-coupled logic. *Fujitsu; free to qualified mutual customers of Fujitsu and Dazix.*

**Reader Service No. 30**

### ■ Hspice integration

The Hspice optimizing simulator now integrates into the Viewsim/SD simulation environment. Tight feedback loops between the two components' A/D simulators allow for interactive, mixed-signal simulation. Hspice generates compatible data for the

Viewtrace drawing package for analog display, output, and analysis. Hspice and its server for Viewsim/SD are available for Sun Sparcstation, DECstation, IBM RS/6000, and VAX/VMS. Meta-Software.

**Reader Service No. 31**

### ■ C-based debugging

The Silos II 90.1 software release is a C-based version of the original Silos logic and fault simulation program. It features two-dimensional interactive debugging, time enhancements that support Future Net-compatible ASIC part libraries, spike simulation, and analog behavioral modeling. Users can create custom reports using the 90.1's software tools. *Simucad; immediate delivery.*

**Reader Service No. 32**

### ■ More midlayers in PCB program

Version 2.0 of the Tango-PCB software offers component placement assistance (manual, interactive, or automatic), polygon fill, graphics support, and the addition of four midlayers for a total of 23 layers. Tango-Route 2.0 includes autorouting capabilities, editable power/ground planes, expanded memory support, and increased maze routing, according to the company. *Accel Technologies; from \$495 (individual entry-level tools) to \$1,695 (Tango-PCB and Tango-Route, available from stock).*

**Reader Service No. 33**

### ■ New wave of Pspice

The Pspice circuit analysis version 4.04 contains an OS/2 real-time waveform viewer that allows viewing of output waveforms while running a simulation. Other enhancements include previously defined global parameters for new expressions, and two library files containing seven amplifier models and more than 400 SCR and triac models. *Micro Sim Corporation; from \$950 to \$29,900.*

**Reader Service No. 34**

### ■ Smart prototype development

Designers using Smartmodels can simulate and verify a system-level software design before entering the hardware prototype phase. Smartmodels interface with Altera Multiple Array Matrix (MAX) erasable, programmable logic devices (EPLD) and tools to ensure accurate modeling of logic delays in the compiled EPLD design. Smartmodels are compatible with a range of systems and workstations. *Logic Automation; \$7,900 (subscription service fee per workstation).*

**Reader Service No. 35**

### ■ Microwave Hspice

A host of microwave components is now available on the H9001, the latest version of the Hspice optimizing analog circuit simulator. These components include lossy transmission lines, geometric GaAs FET (gallium arsenide, field-effect transistor) with backgate, quasisaturation BJT (bipolar junction transistor), and submicron metal-oxide semiconductor FET). The H9001 also supports algebraic expressions, multiple simulation viewing, and a user interface for X Windows and Postscript. *Meta-Software; immediate availability.*

**Reader Service No. 36**

### ■ Mixing text and CAD graphics

Engineers and architects can attach part, size, and cost information to CAD elements by using the Versadata/386 package. A designer can visually verify drawing elements to which information is attached as well as sort and tally this information to generate reports. Versadata/386 requires a dual-screen configuration for displaying graphics and text; a VersaCAD/386 version 5.4, revision 7 or greater; and at least 4 Mbytes of memory. *Computervision; \$995.*

**Reader Service No. 37**

### ■ 3-D modeling

The Betasoft-R software program analyzes electronic boards for thermal

reliability. It performs three-dimensional modeling on the complex flow and thermal fields by evaluating heat conduction, convection, and radiation. The menu-driven program outputs thermal maps of a board and its components. Betasoft-R also comes with a library of 2,500 components, and it operates on any XT/AT/PS2-compatible computer using DOS 3.0 or higher. *Arctos Systems Corporation; \$2,395 (model R-1S, single-sided), \$3,595 (model R, double-sided).*

**Reader Service No. 38**

## Computers and keyboards

### ■ Scanners provide two options

Two bar code scanners collect data by different methods. The Dynabar-232 scanner plugs into a Psion hand-held computer, enabling data collectors to scan with one hand. The result is a combination bar code reader, terminal, and RS-232 communications link.

For industrial, retail, and office applications with critical power consumption needs, the Welch Allyn bar code model offers a press-to-operate switch that directly controls power to the scanner. *Psion.*

**Reader Service No. 39**



*Dynabar-232 bar scanner*

### ■ Six microterminals for OEMs

The 100 and 200 series of CTM microterminals provides an operator interface or a control panel solution for original equipment manufacturers. The



CTM150 and 170 models come in transistor-transistor-logic-level RS-232 or multidropped RS-422 ports. The CTM200 (with RS-232) and CTM220 (with RS-422) allow access to 16 functions defined by the user. Models 250 and 270 combine backlit function keys, internal beeper, and 32-pin connector with the standard features of the CTM200 and 220. *Burr-Brown Corporation*; from \$195 (CTM150) to \$295 (CTM270), available from stock.

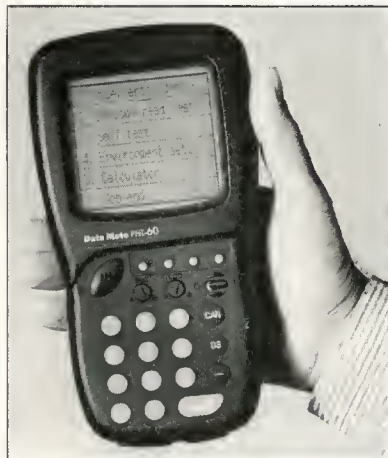
**Reader Service No. 40**

### ■ One-handed data collection

Featuring a numeric keypad, programmable function keys, and a touch-panel liquid crystal display, the Data Mate PHT-60 hand-held terminal is available with 256 or 512 Kbytes of memory.

The LCD allows the viewing of 6 lines  $\times$  24 characters, or 128  $\times$  192 pixels in graphics mode. The terminal reads four bar code types (Code 39, UPC/EAN/JAN, 2 of 5, and Codabar) and communicates with the host computer via a RS-232C serial port. *Timekeeping Systems*.

**Reader Service No. 41**



*Data Mate PHT-60 terminal*

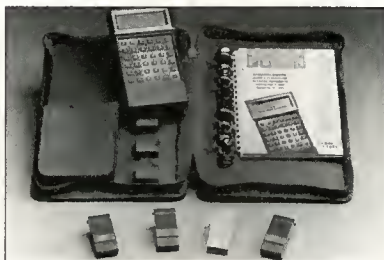
### ■ Hand-held computer briefcase

The Executive Hand-Held Computer Kit unzips to reveal a computer and

applications software on one side, and a personal diary/calendar binder on the other. The IBM-compatible LZ terminal contains 32 Kbytes of internal RAM and 64 Kbytes of internal ROM.

The kit includes a Lotus 1-2-3-compatible spreadsheet, financial programs, eight games, and custom design capabilities. It also comes with a 64-Kbyte EPROM (erasable, programmable, read-only memory) data package for off-line storage. *Psion*; \$720.

**Reader Service No. 42**



*Executive Hand-Held Computer Kit*

### ■ Low-power EL displays for medical applications

Solid-state electroluminescent (EL) displays now offer power consumption ratings comparable to LCDs. A redesigned Planar model requires as little as 3.6 watts for a typical image of waveforms and text found in medical instrumentation. The EL displays offer twist-tab packaging. *Planar Systems*.

**Reader Service No. 43**

### ■ Touch screen dynamics

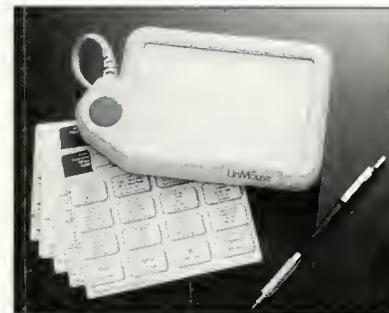
A flat-panel monitor includes an EGA-compatible (extended graphics adapter) electroluminescent display and a high-resolution touch screen. According to the company, the El Touch's analog screen reads user contacts with the solid-glass sensor even with a dirty screen. The monitor features a display area of 8 by 5 inches and measures 3 inches deep. It plugs into an EGA card and requires no modifications to software. *Micro Touch*; \$1,995 or \$1,495 (OEM).

**Reader Service No. 44**

### ■ Alternative mouse device includes stylus option

A touch-sensitive tablet called the Unmouse provides another option for mouse and trackball users. Running one's finger along the 3  $\times$  4-1/2-inch tablet moves the cursor, and pressing lightly simulates the clicking of a mouse button. Used as a function keypad, the Unmouse comes with templates that slip under the glass tablet to emulate PC function keys. Word Perfect and Lotus 1-2-3 templates are included, as well as blank templates and software for custom keystroke programming. In the Unmouse's Absolute mode, users can draw on the tablet with a stylus. The Unmouse is compatible with IBM machines. *Micro Touch*; \$235.

**Reader Service No. 45**



*The Unmouse tablet*

### ■ Software emulates terminals

An emulator for PCs running under MS-DOS, the Zstem 220 software provides terminal emulation on PW2s, PCs, XTs, ATs, PS/2s, and compatibles. It also emulates a DEC VT220 terminal with VT320 enhancements such as status line, International Standards Organization characters, and screen saver. The Zstem 220 software includes error-free file transfer protocols; printer, plotter and network drivers; and a script language. Zstem 220 is available from Unisys under its Desktop III contract with the US Department of Defense, or call KEA for ordering information. *KEA Systems*.

**Reader Service No. 46**

What's #1 in  
the hearts of  
its readers?

**IEEE Micro!!**

How does  
it do it?

**With quality  
articles!**

Who keeps  
the quality  
high?

**The article  
reviewers!**

**Become a reviewer and  
join the #1 team.**

Want to help keep *IEEE Micro* #1? Editor-in-Chief Joe Hootman seeks more technical reviewers—people interested in seeing that the articles published in *IEEE Micro* continue to be of the highest quality. The technical review process is a crucial step in providing readers with correct and timely information so they can keep up with their ever-changing profession.

Send your professional  
information directly to:  
Joe Hootman  
University of North Dakota  
PO Box 7165  
Grand Forks, ND 58202

**Single-board computers and  
controllers**

■ **Microcontroller allows  
added access**

Measuring 3-1/2 × 4-1/2 inches, the Control R II 8031-based microcontroller module allows access to the address, data, and control bus through newly added expansion headers. The module also incorporates space for 8 Kbytes of on-board static RAM and jumpers that allows for alternate processors using ROM languages. The unit supports applications such as data logging, lab instruments, robotics, motor and power control, and remote monitoring. *Sintec*; \$64.95.

**Reader Service No. 47**

■ **One-slot board for  
backplanes**

Slot spaces are usually at a premium for standard PC/AT bus passive backplanes. Original equipment manufacturers can install the Slot Board/386, which occupies one slot instead of several. The board includes a 20-MHz, 32-bit 80386 CPU with a maximum of 4 Mbytes of RAM as well as I/O and disk controllers. Its design serves embedded applications such as point-of-sale terminals, medical instruments, machine control, and diskless workstations. *Ampro Computers*; \$1,170 OEM (100s); available 30 days ARO.

**Reader Service No. 48**

■ **Heating up clock rates**

Ice Cap 486 achieves higher clock rates by lowering the microprocessor's operating temperature to 0° Celsius and controlling the voltage. The ICEC (integrated circuit, environmentally controlled) component houses this cooling technology, allowing a 38-MHz chip to register a Landmark Benchmark rating of 170, according to the company. The Ice Cap 486 operates at 30.6 MIPS (million instructions per second) at 38 MHz. *Velox Computer Technology*; \$150 OEM (without microprocessor).

**Reader Service No. 49**

■ **Axis management with  
motion controller**

The SRX motion controller manages a maximum of eight axes and works with a range of computers through the RS232 or RS422 serial port. It also communicates with PLCs (power-line carriers) through 9 bits of user-defined I/O lines. Motor drives for stepping, linear, or servocontroller functions control the number of axis configurations. Optimized for microstepping, the SRX controls high-resolution motors up to 50,000 steps per revolution with an appropriate driver and motor. *Oregon Micro Systems*; from \$225 (per axes).

**Reader Service No. 50**

■ **Compact AT-size  
motherboard**

The 13.3 × 8.5-inch K386SX motherboard accepts up to 8 Mbytes of dynamic RAM in a combination of 256-Kbyte and 1-Mbyte modules. The AT-size board includes an Intel 386 SX microprocessor, floppy disk controller, and LIM (Lotus/Intel/Microsoft/AST Research expanded-memory specification) 4.0 support. Users can select 8-MHz or 16-20-MHz CPU speeds from the keyboard. The K386SX board performs at zero wait states with 80-ns DRAMs for 20 MHz. *Klever Computers*; \$360 (from two to 25 units).

**Reader Service No. 51**

**Reader Interest Survey**

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 180    Medium 181    High 182



# Product Summary

Joe Hootman

University of North Dakota

Manufacturer	Model	Comments	R.S.#
<b>Memories</b>			
Accutec Microcircuit Corp.	AK594096, AK584096	Compact module series features 4-Mbyte DRAM density for 8-bit data paths in either SIMM or SIP versions with 60- and 100-ns access times. A configuration for applications requiring the ninth bit for parity is available.	80
International Microelectronic Products	IMP234XX	Family of 4-Mbit, JEDEC ROMs stores PC word processor and spreadsheet applications. Organized as 512K × 8 bits or 256K × 16 bits, the battery-powered, CMOS devices access data in 110 ns to support microprocessors without wait states. From \$4.75 each (5,000s).	81
Philips Components-Signetics	48F010	One-Mbit, byte-wide Flash memory features sector and chip erase and overerase protection. Erasable in bulk or in 128 increments, the JEDEC device supports start-up functions and system operation by keeping a section of code intact. \$15 each (1,000s).	82
Raytheon Company Semiconductor Div.	R29771, R29773	Standard bipolar PROM and power-switched SPROM replace the R29671 and R29673 devices. The 55-ns, 4,096 × 8 devices come in commercial and military temperature-range versions. \$9 each (100s).	83
<b>Processors/controllers</b>			
Intel Corporation	80C186EB	Embedded processor designed as a CPU for mobile applications such as cellular phones retains its status during power-down modes. Code-compatible with its 8086 and 80186/80C186 predecessors, the 16-bit unit also controls data in office automation products and communications and industrial control applications. \$16.95 each (1,000s).	84
Mesa Electronics	6P21	PC bus-compatible coprocessor fits on one 4.2 × 5.5-inch card for use in I/O-intensive and real-time control applications. Hosts communicate through a 1-Kbyte, dual-ported RAM. The 96-Kbyte RAM slave processor remains independent of the host bus except for communications. Each unit requires 2 Kbytes of host memory space for communication and one host interrupt line. \$335 each.	85
Metheus Corporation	1100/1200 series	Graphics controllers perform 10 million-pixel/s random vector draw and polygon fills in excess of 60 million pixels/s. The 1100 line offers 1,024 × 768 resolution, 4- or 8-bit planes, up to 256 displayable colors from a palette of 16.7 million, VGA emulation, and up to 1 Mbyte of on-board memory. The 1,280 × 1,024, noninterlaced monitor support of the 1200 series features up to 2 Mbytes of on-board memory and VGA pass-through. From \$1,999.	86

## Advertiser/Product Index

IEEE Computer Society Membership .....	98
IEEE Computer Society Press .....	102
Microway .....	Cover IV
STN International .....	1

### FOR DISPLAY ADVERTISING INFORMATION, CONTACT:

**Northern California and Pacific Northwest:** John D. Vance & Associates, Inc., 4030 Moorpark Ave., Suite 116, San Jose, CA 95117; (408) 741-0354.

**Southern California and Mountain States:** Richard C. Faust Co., 24050 Madison St., Suite 101, Torrance, CA 90505; (213) 373-9604.

**Midwest:** The Kingwill Company, 4433 W. Touhy Ave., Suite 540, Lincolnwood, IL 60091; (708) 675-5755.

**East Coast:** Atlantic Representative Group, 349 Maple Place, Keyport, NJ 07735; (908) 739-1444.

**New England:** Arpin Associates, P.O. Box 6444, Holliston, MA 01746; (508) 429-8907.

**Europe:** Heinz J. Görgens, Parkstrasse 8a, D-4054 Nettetal 1 - Hinsbeck (F.R.G.); phone: (0 21 53) 8 99 88; fax: (0 21 53) 8 99 89.

**Southwest/Southeast:** Heidi Rex, Office, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; (714) 821-8380.

For production information, conference, and classified advertising, contact Heidi Rex or Marian Tibayan.

**IEEE MICRO**, 10662 Los Vaqueros Cir., PO Box 3014, Los Alamitos, CA 90720-1264; phone (714) 821-8380; fax (714) 821-4010.

	RS #	Page #
Accel Technologies	33	56
Accutek Microcircuit Corp.	80	59
ADAC Corp.	13	53
Alligator Technologies	15	54
Ampro Computers	48	58
Arctos Systems Corp.	38	56
Ariel Corp.	18, 19	54
BSO/Tasking	12	53
Burr-Brown Corp.	10, 11, 40	53, 57
Computervision	37	38
Contec Microelectronics USA	28	55
Data Translation	14	54
Electrical Engineering Software	26	55
Fijitsu	30	55
Intel Corp.	84	59
International Microelectronic Products	81	59
KEA Systems	46	57
Klever Computers	51	58
Logic Automation	35	56
Mesa Electronics	85	59
Meta Software	31, 36	56
Methus Corp.	86	59
Micro Networks	23	55
Micro Sim Corp.	34	56
Micro Touch	44, 45	57
National Semiconductor	20	54
Olsson Engineering	22	55
Oregon Micro Systems	50	58
Philips Components-Signetics	82	59
Planar Systems	43	57
Psion	39, 42	56, 57
Rapid Systems	16, 24	54, 55
Raytheon Company	83	59
Real Time Devices	25	55
Rhetorex	17	54
Simucad	32	56
Sintec	47	58
STN International	1	1
Tatum Labs	27, 29	5
Tetra Systems	21	54
Timekeeping Systems	41	57
Velox Computer Technology	49	58

## Moving?

PLEASE NOTIFY  
US 4 WEEKS  
IN ADVANCE

\_\_\_\_\_  
Name (Please Print)

\_\_\_\_\_  
New Address

\_\_\_\_\_  
City

\_\_\_\_\_  
State/Country

\_\_\_\_\_  
Zip

### MAIL TO:

IEEE Computer Society  
Circulation Dept.  
PO Box 3014  
10662 Los Vaqueros Circle  
Los Alamitos, CA 90720-1264

ATTACH  
LABEL  
HERE

- This notice of address change will apply to all IEEE publications to which you subscribe.
- List new address above.
- If you have a question about your subscription, place label here and clip this form to your letter.



## Advanced information processing

*continued from p. 15*

fied that it was only necessary to take the process as far as designing a parallel architecture and simulating it.

The subproject advanced the state of the art in implementing lazy functional languages in a number of ways. It defined a parallel model that naturally extends the sequential lazy evaluation and preserves the semantics of programs. The parallelism information for the model can be determined in the compiler using an analysis technique called abstract interpretation. We've proved that when the information obtained from the abstract interpretation is used, we retain the semantics of the program. We implemented the analysis technique in another project.

We based the parallel execution model strongly on lazy evaluation, with some extra features that create subprocesses to perform some of the computation. Therefore, the design of a parallel machine fell into two fairly distinct halves: a machine to support the sequential core of the language, and a parallel harness that supports the specifically parallel parts of the evaluation model. We designed a sequential abstract machine, an improvement on extant abstract machines. We then added the extra features determined in Burn<sup>8</sup> to turn it into an abstract distributed memory architecture, showing how to compile code for it.

We established that most features of functional languages can be implemented on current hardware, thus avoiding the need to design any specialized hardware. We also prepared a demonstration implementation on a transputer network and translated the parallel abstract machine code into transputer machine code.

Functional languages, logic, and object-oriented languages require automatic storage allocation and deallocation (garbage collection). Subproject B developed a garbage collection algorithm<sup>3</sup> and subsequently improved it.

The efforts in this subproject have led to a complete system, from a functional program without parallel constructs to an implementation on a parallel machine. This exceeds the original goal.

**The logic database approach.** The main objective in subproject C was to design a parallel deductive database machine called the Delta Driven Computer.<sup>9</sup> The DDC is specialized to execute deductive requests in parallel by using relational operations. All relations reside in main memory.

We based the execution model on an original technique called the Alexander method,<sup>10</sup> which was designed to merge recursive views in queries. The advantage of this merge operation is that it produces a set of rules that can be executed in a forward-chaining strategy without computing useless information. A compilation technique forms the basis for this method rather than other techniques based on interpretation.

The advantage is that following the execution model, the set of rules can further be compiled into a low-level language in which the parallelism is explicit. Each node of the DDC machine can directly execute this language, and two intermediate languages (Vim and DDCL).

The DDC machine organization consists of four to 256 identical nodes connected by a network. It is a "shared-nothing" architecture in the sense that no shared memory exists. Message-passing techniques accomplish all communications between the nodes. Each DDC node contains a general-purpose processor, a special coprocessor called Music, a large memory space, and a communication device. The Music coprocessor speeds up relational operations (its development was partially financed by ESPRIT project 956 and a national project).

DDC is currently simulated on a four-processor Unix machine (SPS7-70). It demonstrates that the compiled approach permits the exhibition of parallel code from declarative languages (SQL and Vim). This simulation very closely resembles the real prototype (eight processors with no shared memory) on which we've started the implementation of the DDC software. The characteristics of both the prototype and the simulation permit us to deduce the performance on the future DDC machine by performance modeling.

The challenge in a parallel shared-nothing architecture is to provide for production of an efficient parallel code. In DDC, we consider this code efficient if the grain of the parallelism is coarse (more than 1,000 instructions between two communications).

**The logic + functional approach.** Different from the other subprojects, subproject D aimed to evaluate the possibility and the advantage of integrating the dominant declarative programming styles, logic (L) and functional (F).

We've designed two languages: a first-order L+F integration, which is both semantically well-defined and efficiently implementable by a single computational mechanism for sequential and parallel implementations. The corresponding language K-Leaf, based on Horn Clause Logic with Equality, extends pure Prolog to express nonterminating, conditional term-rewriting systems with constructors.

We defined Ideal (an Ideal DEductive and Applicative Language), the first compiled, higher-order L+F language, as a user language of very high level offerings. Beside the usual Prolog capabilities, it offers the most distinctive features, now present in modern functional languages: lazy evaluation, type inference, and higher order constructs.

Next, we devoted effort to the efficient implementation of the integrated computational model on generic sequential processors and on parallel architectures. We designed a conservative extension of the WAM (Warren Abstract Machine) called K-WAM, to efficiently implement outermost resolution. K-WAM, based on the extensive experience available for sequential and parallel Prolog compilation, is suited to

incorporate all the new incoming optimizations. Extensions to support the dynamic resolution strategy required by K-Leaf thus allowed the first compiled implementation of a sound L+F language.

A strong synergy with ESPRIT project 26, which covered the architectural aspects, enabled the implementation of an L+F language on a real parallel machine. The machine embodies three main concepts: physically distributed/logically shared memory, very fast context switching built into the processing element, and support of efficient packet-switched, nonlocal communications. The machine is an ensemble of up to 128 transputers fully interconnected by a low-latency (2- $\mu$ s transit time), high throughput (10 Mbytes/s on each port), "cut-through" packet-switching, and a buffered Delta network implemented by means of a custom VLSI switching element circuit containing about 30,000 CMOS (complementary metal-oxide semiconductor) devices.

We found parallel execution models for L+F languages to be viable on distributed-memory architectures. For efficiency reasons, the programmer controls the granularity of parallelism, through special annotations and setlike constructs in a disciplined way. We prototyped a (deterministic And)-Or parallel model reduction. We finally ported a restriction of it, (independent And)-Or parallelism, to our parallel machine. We accomplished the porting by means of an original mapping of And to Or parallelism, as, independently, experimented with success in the Giga-Lips Project.

We assembled most of the above-mentioned results in a working prototype of a small-configuration machine (16 processing elements). This machine runs conventional benchmarks (N-queens, Fibonacci numbers) and a few more realistic applications (grammar-based image recognition, logic simulation/fault finding). Written in And-Or parallel K-Leaf, they show quasilinear speed-ups at 75-90 percent of the ideal efficiency.

The measured performance of the mapping sequential K-WAM into C, on standard benchmarks is:

- 80-300 percent of Quintus Prolog,
- 50-200 percent of G-Machine-based Lazy ML, and
- 20-80 percent of C used with all of its imperative features.

This experience suggests that a reasonably good performance (160 KLIPS—thousands of logical inferences per second—on a Sun 3/280) can be obtained.<sup>11</sup>

**The dataflow approach.** The use of the dataflow principle for parallel execution has a long history. Since the early seventies, researchers have proposed a large number of dataflow computer architectures, and some of them have even been built, for example, the Manchester Data Flow Machine at Manchester University. But most of these systems exploit fine-grain parallelism and run number-crunching programs,

thus leading to expensive special-purpose hardware. In contrast, subproject E at Stollmann GmbH intended to build a dynamic dataflow machine suited to run commercial applications and to exploit coarse and coarsest granularity.

Stollmann chose a database application, the parallel evaluation of database queries. Because of our concept of variable granularity, designers could deviate from a highly specialized hardware approach and start by building a demonstrator with off-the-shelf hardware. The dataflow control mechanisms are lifted to the software level, including distributed parallel firing. The basic principles of the approach are load-distribution mechanisms aiming at exploitation of locality and dynamic load-balancing mechanisms such as task attraction.

The designers formed an abstract architecture model, the Stollmann Data Flow Machine. The SDFM units act as software processes. The execution model consists of three different units. Execution units execute the fired dataflow nodes, that is, the executable instructions, and produce output operands. The firing control units assign the produced output operands to the corresponding instructions and fire the executable instructions. The administration control unit performs a special function; it monitors the system, starts and terminates programs, and executes input and output instructions.

These units communicate asynchronously via buckets, a special kind of queue with a form of intelligence and access strategies. The buckets distribute messages to exploit locality; the work must be done by the unit in which the corresponding data are stored. Moreover, the system provides dynamic load balancing to prevent exploitation of locality to lead to nonoptimal load distribution. When a unit runs out of work, it attracts work from other units.

In programming the SDFM, designers used the Blass and Clan languages. A program written in Blass can be seen as the textual representation of a dataflow graph. Clan is a functional, single-assignment language similar to Sisal. To support coarse-grain dataflow, designers introduced the concept of user-defined instructions into both languages.

We implemented the SDFM model on our demonstrator, a multiprocessor configuration with four processor boards connected by a VMEbus. For the first implementation we chose 68020 processor boards equipped with dual-ported RAM, thus admitting a global address space. One processor board runs Unix Version 3 and acts as a host. The other boards run Srtx, a real-time operating system kernel with multitasking and dynamic memory management. Smocs, a common layer to all processor boards, provides a basic set of operating system functions for a multiprocessor system, especially global queues and management of the local, but shared, memory.

Clan realized our application, the parallel evaluation of database queries, by implementing the relational algebra operations as user-defined instructions. These operations



**Table 3. Languages and architectures of the subproject systems.**

Subproject	Language	Architecture	No. of nodes
A	POOL*	Pooma: Packet-switching	100
B	Miranda	emulation on a transputer network	4
C	SQL, Vim*	DDC: Bus	8
D	Ideal*	Transputer, multistage network	16
E	Sisal, Clan*	SDFM: Bus emulation	4
F	Lop*	SDFM: Bus emulation	4

\*New language

processing complete relations split into parallel package versions to introduce more parallelism. Our performance measurements show that speed-up depends largely on the grain size. An acceptable speed-up can be gained by a grain size of 0.1 seconds. Though there is no one bottleneck, optimization of the system to reach an optimal grain size of 0.05 seconds is possible as is even finer grains by introducing special hardware.

**The connection approach to logic programming.** The main goal of subproject F was an inference machine based on the parallel, automated theorem prover Partheo<sup>3</sup> for full first-order predicate logic. The underlying proof calculus is model elimination, a specialization of the connection method developed by W. Bibel. The input language Lop for Partheo allows a straightforward declarative style of logic programming without being restricted to Horn Clause logic as in Prolog.

The second goal of subproject F was to improve and implement the functional parallel programming language called FP2. FP2 provides parallel processes based on term rewriting and communication via unification. It is used as a high-level specification tool for the parallel algorithms used in Partheo.

LIFIA, in Grenoble, France, developed the language FP2 and provides the sequential implementation of FP2. The group at the Technical University of Munich worked on design and implementation of Partheo and Lop. The Nixdorf part of the project covered the parallel implementation of FP2, some contributions to the work on Partheo, and exploitation of project results.

Setheo, the Sequential Theorem Prover, extends the Warren Abstract Prolog Machine to full first-order predicate logic. Setheo, implemented in C, yields a performance of about 120 KLIPS on a Sun 4 machine. On Unix machines, Setheo proved to be one of the fastest existing, high-performance theorem provers as well as an efficient Lop interpreter.

Partheo, the Parallel Theorem Prover, runs on a network of 16 transputers. It solves independent parts of the search

tree in parallel. Together with the fast abstract machine of Setheo, the parallel prover increases the performance of the inference machine. An easy-to-use graphical user interface facilitates the development and testing of Lop programs.

A sequential FP2 interpreter running on Sun workstations represents a powerful high-level language for specification, verification, and test of parallel algorithms.

We implemented a high-performance FP2 interpreter<sup>4</sup> on a parallel VMEbus machine (the Stollmann test machine built by subproject E).

**An evaluation.** When project 415 finished in 1989, each of the subprojects had delivered a prototype parallel system, executing programs in a language for the particular programming style. Table 3 summarizes these results. The design of a number of new languages (indicated by the asterisk in the table) reflects the improved understanding of the semantics of concurrency and its inclusion in the various symbolic programming styles.

We knew the models of computation for sequential symbolic languages were quite different from imperative languages. Therefore we assumed efficient implementations would require special hardware support, such as special CPUs or dedicated coprocessors. We directed quite some efforts to the design of abstract operational models to support the semantics work and to guide the implementations. Interestingly, a good similarity exists in these models, which view a program as a collection of processes and employ message passing as the means to achieve communication and synchronization. A process can implement an object (as in POOL) or a reduction task that evaluates function-arguments (as in Miranda). Processes and their communication patterns are either statically determined or dynamically created (POOL, Miranda, Lop). The latter case requires a runtime system to manage the allocation and scheduling of the required resources.

Thus we could cleanly separate the issues of parallelism and of symbolic execution (within the otherwise sequential

processes). Our efforts led to several novel compilation techniques that result in efficient mappings of the diverse high-level programming languages on standard von Neumann microprocessor architectures, thus falsifying the hypothesis on special hardware support. Our implementation of POOL on sequential machines, for example, approaches the performance of an equivalent C program within a factor of two.

This important conclusion means that parallel symbolic computers can directly benefit from the technology improvements in standard microprocessors, and that major investments in VLSI development for processors are not necessary.

A second conclusion is that the operational models alleviate notions of sharing. Thus they match a loosely coupled architecture, which may consist of self-contained computers augmented with communication facilities. This also means that we restrict extra hardware facilities to the support of communication between processes.

End-to-end routing devices that guarantee deadlock-free operation in sparsely connected topologies form the major innovation. These devices live with the dynamic creation of processes and communication patterns, and offload the transmission overhead from the CPU. The Pooma communication processor employs a packet-switching method and distinguishes itself from other devices (chiefly, the iPSC) by very efficiently handling short messages between objects.<sup>12</sup>

At least two areas of research to further improve the efficiency of execution deserve further attention. First is the issue of granularity. Related to the nature of symbolic data and operations, the processes tend to be very lightweight (several orders-of-magnitude smaller than in Unix) and require inexpensive mechanisms for scheduling and resource allocation. Second, when many processes reside on the same node, a "postman" device could place incoming messages in the right queues and perform the administration tasks. This device would reduce communication costs further.

Our final observations relate to the parallel applications that were investigated in the project. They indicate that parallel implementations of symbolic algorithms can be expressed very well in the newly designed languages, and that we can achieve good speed-up, absolute performance, and expandability.

## Pooma

The following describes in more detail the developments in ESPRIT subproject A. This subproject included the parallel object-oriented language POOL,<sup>13</sup> the parallel object-oriented machine Pooma; and applications written in this language and running on this machine.<sup>14-17</sup>

POOL allows the programmer to express parallelism explicitly by specifying objects that run in parallel with each other and communicate via messages. Note that this work has been carried out in a close cooperation between Project 415-A and Prisma, a Dutch national project. Philips Research

Laboratories worked together with the Centre for Mathematics and Computer Science in Amsterdam and the universities of Amsterdam, Leiden, Twente, and Utrecht.<sup>15</sup>

Pooma is a machine with a loosely coupled MIMD (multiple-instruction, multiple-data) architecture.<sup>18</sup> It is composed of many nodes, each of which consists of a data processor, memory, a communication processor, and I/O devices. The communication processors interconnect with serial, bi-directional, point-to-point connections (links). The network of communication processors provides an end-to-end, deadlock-free, packet-delivery service to the data processors of Pooma.

The POOL language model and the Pooma machine model correspond closely. Each node executes a number of objects, which communicate by messages that are sent over the network of communication processors in the form of packets. A compiler and an operating system (runtime support) map a POOL program onto the architecture.<sup>19</sup> The operating system directly supports processes and messages as POOL requires.

We aimed at symbolic processing, numerical computing, and data-intensive applications such as databases and document retrieval systems. Currently we've placed the most emphasis on data-intensive applications for the office environment.<sup>14</sup>

## Language

We directed the effort in language design at exploiting the ideas of object-oriented programming to support the efficient programming of highly parallel systems. We did not intend the resulting language (POOL) to be a tool for rapid prototyping but a language for the systematic construction of large, reliable systems. We outline POOL here.

**Objects.** In object-oriented programming, programmers view an executing program as a collection of *objects*. An object is an integrated unit of data and procedures that can act on these data. *Variables* store data; the procedures belonging to an object are called *methods*. Each variable can store a reference to an object. The data of an object are not directly accessible to any other object. Objects can only interact by exchanging *messages*.

A special characteristic of POOL is that, as soon as an object is created, it starts executing its *body*; a local process. Different objects execute in parallel and may interact by explicit sending and answering of messages. Within an object all activities occur sequentially; having parallelism inside objects would cause problems by necessitating additional mechanisms to synchronize processes.<sup>13</sup>

**Classes.** Objects are created by other objects. An object can modify its own data and can have its own independent internal activity. To describe this unbounded number of dynamic objects in a static, finite program, the programmer defines objects by grouping them into *classes*. All the objects in one class (the *instances* of the class) have exactly




**December 1990 issue** (card void after June 1991)

Name \_\_\_\_\_

Title \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Country \_\_\_\_\_ Phone( \_\_\_\_\_ ) \_\_\_\_\_

**Please send** (Circle those you want)

- 201** Publications catalog  
**202** Membership information  
**203** Student membership information  
**204** Senior membership information  
**205** IEEE Micro subscription information

**Reader Interest**  
 (Add comments on the back)

Readers,  
Indicate your interest in  
articles and departments  
by circling the appropriate  
number (shown on  
the last page of articles  
and departments):

150 151 152 177 178 179  
153 154 155 180 181 182  
156 157 158 183 184 185

159 160 161 186 187 188  
162 163 164 189 190 191  
165 166 167 206 207 208

168 169 170 209 210 211  
171 172 173 212 213 214  
174 175 176 215 216 217

**Product Information** **1**  
 (Circle the numbers to receive  
product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200


**December 1990 issue** (card void after June 1991)

Name \_\_\_\_\_

Title \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip \_\_\_\_\_

Country \_\_\_\_\_ Phone( \_\_\_\_\_ ) \_\_\_\_\_

**Please send** (Circle those you want)

- 201** Publications catalog  
**202** Membership information  
**203** Student membership information  
**204** Senior membership information  
**205** IEEE Micro subscription information

**Reader Interest**  
 (Add comments on the back)

Readers,  
Indicate your interest in  
articles and departments  
by circling the appropriate  
number (shown on  
the last page of articles  
and departments):

150 151 152 177 178 179  
153 154 155 180 181 182  
156 157 158 183 184 185

159 160 161 186 187 188  
162 163 164 189 190 191  
165 166 167 206 207 208

168 169 170 209 210 211  
171 172 173 212 213 214  
174 175 176 215 216 217

**Product Information** **2**  
 (Circle the numbers to receive  
product information)

1	21	41	61	81	101	121	141
2	22	42	62	82	102	122	142
3	23	43	63	83	103	123	143
4	24	44	64	84	104	124	144
5	25	45	65	85	105	125	145
6	26	46	66	86	106	126	146
7	27	47	67	87	107	127	147
8	28	48	68	88	108	128	148
9	29	49	69	89	109	129	149
10	30	50	70	90	110	130	—
11	31	51	71	91	111	131	—
12	32	52	72	92	112	132	192
13	33	53	73	93	113	133	193
14	34	54	74	94	114	134	194
15	35	55	75	95	115	135	195
16	36	56	76	96	116	136	196
17	37	57	77	97	117	137	197
18	38	58	78	98	118	138	198
19	39	59	79	99	119	139	199
20	40	60	80	100	120	140	200

# SUBSCRIBE TO IEEE MICRO

## All the facts about today's chips and systems

☐ **YES, sign me up!**

If you are a member of the Computer Society or any other IEEE society,  
pay the member rate of only \$21 for a year's subscription (six issues).

Society: \_\_\_\_\_

IEEE membership no: \_\_\_\_\_

Society members: Subscriptions are annualized. For orders submitted March through  
August, pay half the full-year rate (\$10.50) for three bimonthly issues.

Full Signature \_\_\_\_\_ Date \_\_\_\_\_

Name \_\_\_\_\_

Street \_\_\_\_\_

City \_\_\_\_\_

State/Country \_\_\_\_\_ ZIP/Postal Code \_\_\_\_\_

☐ **YES, sign me up!**

If you are a member of ACM, ACS, BCS, IEE (UK), IEEE (but not a  
member of an IEEE society), IECEJ, IPSJ, NSPE, SCS, or other qualified  
professional technical society, pay the sister-society rate of only \$38 for a  
year's subscription (six issues).

Organization: \_\_\_\_\_

Membership no: \_\_\_\_\_

☐ Payment enclosed

DC residents: add sales tax

☐ Charge to ☐ Visa ☐ MasterCard ☐ American Express

Charge-card number \_\_\_\_\_

Expiration date \_\_\_\_\_

Month \_\_\_\_\_ Year \_\_\_\_\_

Prices valid through 12/31/91  
12/90 MICRO

Charge orders also taken by phone:  
(714) 821-8380 8 a.m. to 5 p.m. Pacific time  
Circulation Dept.  
10662 Los Vaqueros Circ., PO Box 3014  
Los Alamitos, CA 90720-1264

## Editorial comments

I liked: \_\_\_\_\_

\_\_\_\_\_

I disliked: \_\_\_\_\_

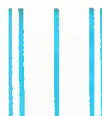
\_\_\_\_\_

I would like to see: \_\_\_\_\_

\_\_\_\_\_

For reader-service inquiries, see other side.

**Reviewers Needed.** If interested, send professional data to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202.



PLACE  
POSTAGE  
HERE

PO Box is for reader-service cards only.

## IEEE Micro

PO BOX 16508  
NORTH HOLLYWOOD CA 91615-6508  
USA



## Editorial comments

I liked: \_\_\_\_\_

\_\_\_\_\_

I disliked: \_\_\_\_\_

\_\_\_\_\_

I would like to see: \_\_\_\_\_

\_\_\_\_\_

For reader-service inquiries, see other side.

**Reviewers Needed.** If interested, send professional data to Joe Hootman, EE Dept., University of North Dakota, PO Box 7165, Grand Forks, ND 58202.



PLACE  
POSTAGE  
HERE

PO Box is for reader-service cards only.

## IEEE Micro

PO BOX 16508  
NORTH HOLLYWOOD CA 91615-6508  
USA



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

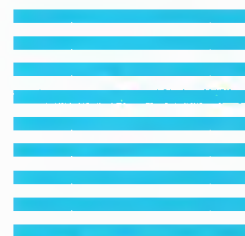
## BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 38 LOS ALAMITOS, CA

POSTAGE WILL BE PAID BY ADDRESSEE

## IEEE COMPUTER SOCIETY

PO BOX 3014  
LOS ALAMITOS CA 90720-9804  
USA





the same structure. That is, they have the same names of variables (though each one has its own set of variables), and they use the same methods to respond to messages. A class definition contains all information that is relevant to the instances of the class and serves as a blueprint for the creation of new instances. POOL provides class parameterization, allowing programmers to define generic classes (for example, a generic class describing arbitrary stacks). Different instances of such a generic class can be created by filling in a concrete class name for the parameters (a stack of integers).

We do not consider classes to be objects themselves, to which one could send messages, as is the case in Smalltalk 80. Nevertheless, we clearly need activities that are associated with a class, instead of with an object (the creation of new objects). Therefore POOL introduces the concept of *routines*. A routine is a procedure that is associated with a class (unlike a method, which is associated with a particular object). Routines can be executed in principle by any object in the system, or even by several objects at the same time.

POOL uses a strong typing mechanism. With each variable a class, its *type*, is associated; it may only refer to objects of that class. Every expression in the language has a type, which indicates the class of objects it can deliver. This capability makes it possible to detect many programming errors before the program is executed and facilitates compiler optimizations.

**Communication.** An object indicates explicitly to which destination object it wants to send a message. Executing the statement `v ! put(56)` sends a message identifying the method (`put`) to the object whose reference is contained in the variable `v`. The message also contains the integer parameter `56`. The sender waits until the receiver processes the message.

Through the statement `ANSWER (put, get)` the receiver indicates that it wants to answer a message that indicates either the `put` or `get` method. The receiver takes the first such message to arrive. If no such message is available when the `Answer` statement executes, the object waits.

Another provision, the *Conditional answer* statement, (for example, `CONDANS (put, get)`) does not block the executing object when no matching message is found. When a message is answered, the object executes the corresponding method, providing it with the parameters in the message. At a certain moment (not necessarily the end), this method returns a result to the sending object, so that this object can continue its activities. The period from the start of the method execution and the returning of the result is called *rendezvous*.

Apart from this *synchronous communication* mechanism, indicated by the `!"` sign—in which the sender waits until the destination returns a result to end the rendezvous—*asynchronous communication* exists, indicated by the `!!` sign. With the latter, the sender does not wait for a result

but immediately continues its own activities. The method, executed by the receiver on accepting the message, does not return a result.

POOL models all program data through objects, even basic data types such as integers and Boolean notations. Message exchanges model all object interactions. To allow for a more compact coding, POOL makes available various syntactic “sugar” constructs (abbreviated notations) to its programmers. For example, the expression `number - 1` is identical to the send-expression `number ! sub(1)`; syntactic sugar constructs exist for most arithmetical notations. The expression `array[out]` is syntactic sugar for the send expression `array ! get1(out)`, which retrieves a value from a one-dimensional array. However, when such a bracket notation occurs at the left-hand side of an assignment, it has a different meaning. The statement `array[in] := e` is equivalent to `array ! put1(in, e)`. This expansion mechanism for syntactic sugar constructs does not only work for basic classes like `Int` and `Array` but also works for programmer-defined classes.

**Units.** A POOL program consists of a series of *units*, which come in pairs of *specification* and *implementation*. An implementation unit contains a number of class definitions. The corresponding specification unit lists a subset of these definitions, which can be used in another unit when the name of the *used* unit appears in the *use list*. Thus the programmer can hide elements explicitly from other units, enhancing abstraction and improving reusability of parts of POOL programs.

SPEC UNIT Bounded\_Buffer

CLASS Buffer(Element)

%% FIFO buffer storing instances of class Element.

ROUTINE new (size : Int) : Buffer(Element)

%% Creates a new buffer which may contain 'size' elements.

METHOD put (e : Element) : Buffer(Element)

%% Adds the element 'e' to the buffer.

METHOD get () : Element

%% Extracts the first element from the buffer.

END Buffer

Figure 1. Specification unit Bounded\_Buffer.

**An example program.** Figure 1 contains a simple, sample specification unit describing a generic class `Buffer`, whose objects function as a bounded buffer of elements of a given

parameter type called Element. Figure 2 presents the corresponding implementation unit, defining the Buffer class.

```
IMPL UNIT Bounded_Buffer

CLASS Buffer(Element)
NEWPAR (size: Int)
VAR cont := Array(Element).new(1, size)
    in := 1
    out := 1
    number := 0

METHOD put (e: Element) : Buffer(Element)
BEGIN RESULT SELF;
    cont[in] := e;
    in := (in // size) + 1;
    number := number + 1
END put

METHOD get () : Element
BEGIN RESULT cont[out];
    out := (out // size) + 1;
    number := number - 1
END get

BODY DO IF    number = 0    THEN ANSWER (put)
    ELIF number = size THEN ANSWER (get)
    ELSE                                ANSWER (put, get)
    FI

OD
YDOB
END Buffer
```

Figure 2. Implementation unit Bounded\_Buffer.

The Newpar clause specifies the parameters for the routine New, which creates new objects of this class. This routine first creates a new object of the class and then sends this object an initializing message containing the new parameters. The first thing the new object does is answer this initializing message. It then initializes the instance variables. After that, the new object starts to execute its body. The body consists of an infinite loop (the loop condition, which is absent here, defaults to true) in which it explicitly answers incoming messages. Depending on whether the buffer is empty (number equals 0) or full (number equals size), or neither empty nor full, the object answers methods put or get, or either. If no such message arrives, the object waits.

Figure 3 indicates how the class Buffer can be used. A POOL program starts its execution by evaluating in parallel the expressions defining the global values of all implementation units. Here the program evaluates the expressions ini-

tializing the three global values prod, cons, and buff in the unit Root, starting an object of each of the classes Buffer, Producer, and Consumer. The Producer object then starts sending a stream of integers to be stored in the Buffer object, which will be emptied by the Consumer object. The program terminates when all activity in the system has ceased. In the example, termination occurs when the consumer has printed all values retrievable from the buffer.

```
IMPL UNIT Root
USE Bounded_Buffer, File_IO

GLOBAL prod := Producer.new()
    cons := Consumer.new()
    buff := Buffer(Int).new(5)

CLASS Producer
BODY
    FOR i TO 100 DO buff ! put(i) OD
YDOB
END Producer

CLASS Consumer
BODY
    DO standard_out ! write_Int(buff ! get() )
    OD
YDOB
END Consumer
```

Figure 3. Implementation unit Root.

## Hardware architecture

As mentioned earlier, the loosely coupled MIMD Pooma machine consists of a network of computing nodes with distributed memory. A packet-switching, point-to-point network allows nodes to communicate. The available communication bandwidth accommodates hundreds of nodes without requiring interconnection links with a very high bandwidth.

**Node architecture.** Figure 4 depicts the structure of a Pooma node. It consists of a data processor, memory subsystem, I/O interface, and communication processor.

The data processor (DP) executes the code of the POOL objects residing on the node. As objects are implemented as control-flow processes, a normal von Neumann architecture suffices for the DP. Each Pooma node can execute several (some 10 to 100) processes. The processor architecture, therefore, supports multitasking, and efficient process switching is a major requirement.

The memory stores the code and data of the operating system (runtime support) as well as the code, stacks, and message queues of the objects residing on the node.



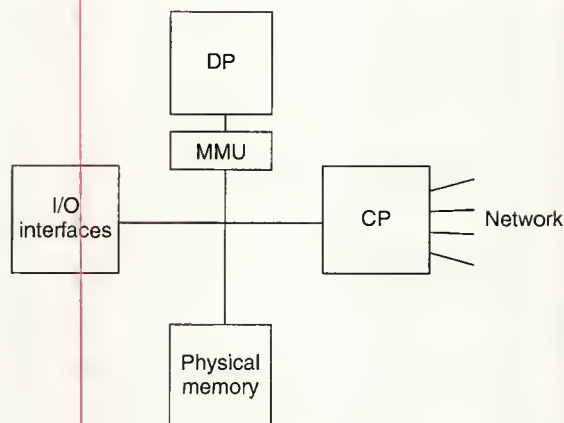


Figure 4. The structure of a Pooma node.

The I/O interfaces, not necessarily available on all nodes, function as LAN and disk controllers. They can be connected directly to the bus of the data processor. We thought this approach to be general enough to handle other types of interfaces in a similar fashion.

Once we decided to connect I/O interfaces to the node bus, we still had further decisions to make. It remains to be decided whether to equip all nodes with I/O interfaces, to provide the possibility on every node to be extended with I/O interfaces, or to equip only particular nodes with actual interfaces or provisions for them. Basically, we face a cost/convenience trade-off here. For inexpensive, standard I/O interfaces, we think it reasonable to provide every node with a SCSI interface to disk or tape unit, serial terminal connections, and even Ethernet). For more expensive interfaces a standard bus extension (VMEbus or Multibus) seems a better solution.

Pooma does not require special facilities such as I/O processors as some data processors can reasonably spend a part of their time servicing I/O interfaces. Every DP can access all I/O system interfaces by passing messages over the network. Since the network provides a very high bandwidth, only its delay can be an obstacle.

**The communication processor.** An important aspect of any decentralized computer architecture is its communication support. The CPs in Pooma together establish a communication system by which every data processor can communicate with every other data processor through packet-switching methods.

Figure 5 shows the way DPs and CPs connect together. Communication takes place in terms of fixed-size, 256-bit packets of information. Each packet contains an 11-bit destination address field. When a source DP sends a data packet to a destination DP, the packet first moves to the local CP via a path of CP-to-CP connections to the CP on the destination

node. Finally it transfers to the destination DP. Each CP uses the destination address, part of the packet, to determine the path through the network. The connection between the DP and the CP is identical to the connection between two neighboring CPs, except that it is implemented as a 32-bit parallel path instead of a serial connection. The CP meets the following requirements:

- *Independent operation.* In particular the data processor need not be involved in the forwarding of packets not yet arrived at their destination.
- *Absence of deadlock and starvation.* The arrival of packets at their destination is guaranteed (provided the destination DP does not stop consuming incoming packets). This means that the CP avoids cyclic wait-for relations between packets.
- *Dynamic and static routing modes.* In the dynamic routing mode packets transfer to a destination via different routes. The main purpose of this dynamic routing is to balance the packet load of the communication network and to optimize use of the large total bandwidth. The disadvantage of dynamic routing is that the order in which a source sends packets to a particular destination and the order in which the destination receives them are not guaranteed to be the same. Use of the communication processor in the static routing mode guarantees order preservation of the stream of packets between source and destination communication processors.
- *Independence of network topology<sup>20</sup> or network size.*
- *Implementable in one VLSI chip.*
- *Efficient usage and administration of packet storage space.*
- *High data throughput.*

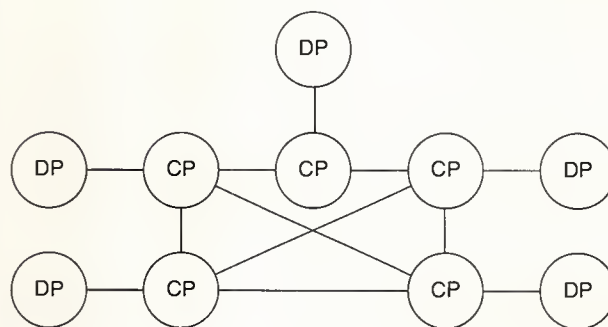


Figure 5. A small instance of the Pooma architecture, consisting of five data processors connected by five communication processors.

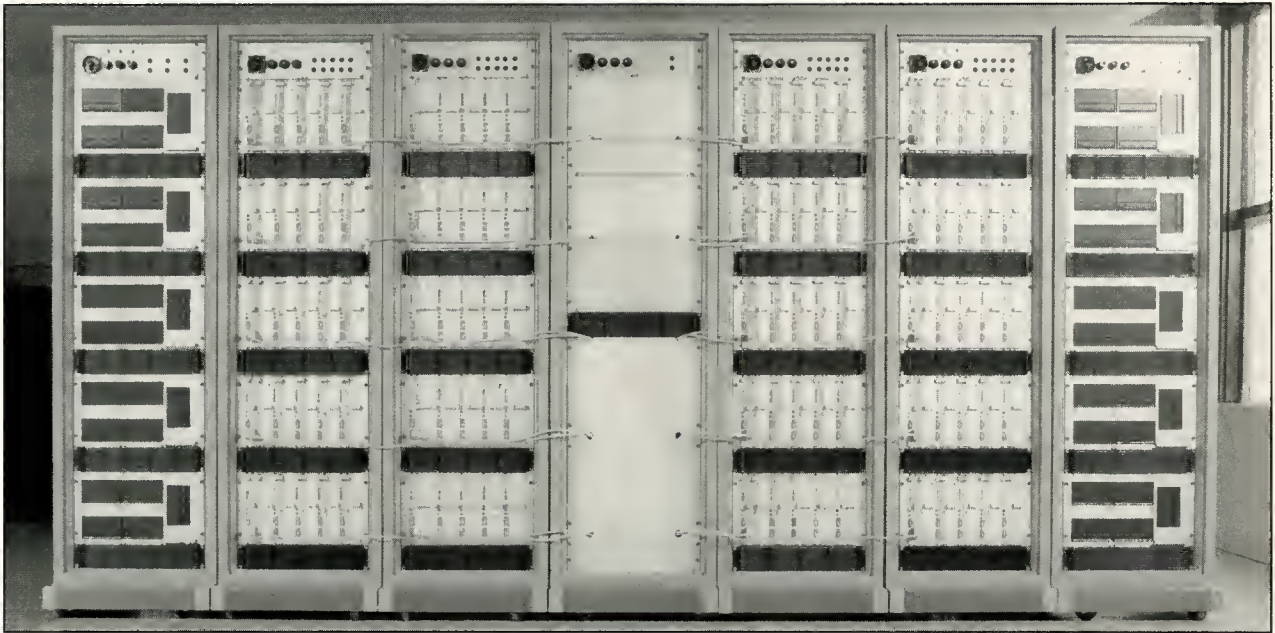


Figure 6. Front view of the Pooma prototype.

The routing function in the CP uses a routing table that is downloaded into the CP upon initialization of the Pooma hardware. When a packet arrives in a CP, the CP checks the routing table with the destination address of the packet to find the routing vector for the packet. This routing vector is a vector of bits, one bit per CP-to-CP output of the CP. Each bit indicates whether or not the packet may be forwarded via the corresponding output. When multiple bits are true, different paths may be taken. When all bits are false, the packet has reached its destination and must be forwarded on via the CP-to-DP output.

A queue of packets accompanies each output of a CP. When a packet arrives in the CP, it resides internally and joins the queues associated with the outputs, which are indicated by the bits in the routing vector of the packet. When an output wants to transmit a packet, it takes the first packet from the queue associated with that output, removes the packet from all queues, and transmits.

The concepts of a routing table and multiple queues together allow the required independence of topology and size of the network and the required dynamic routing. The internal packet storage of a CP contains 255 buffers in each of which one packet can be stored. To meet the requirement of absence of deadlock and starvation, we introduced a new strategy called class climbing, which Annot<sup>21</sup> describes in detail. The nodes of the current Pooma prototype contain a breadboard-version of the CP. Currently, we are developing a VLSI version of the CP.

**The 100-node prototype.** We constructed the Pooma prototype in a very modular way using standard construction materials (see Figure 6). Seven cabinets house the whole machine. The machine contains 50 disks, which occupy two cabinets on the outer sides. Each disk contains its own SCSI interface for connection to the processor board of a node. This disk organization makes the Pooma prototype extremely suitable for data-intensive applications like databases and document retrieval systems. It has a large background memory capacity (total 15 Gbytes), and a high bandwidth-to-background memory (total 50 Mbytes/s). Due to its decentralized nature, a lot of redundancy allows for fault-tolerant designs.

Four cabinets house the 100 nodes. One cabinet contains five crates and each crate contains five nodes. The hardware of a node is built around the VMEbus, as seen in Figure 7. The Motorola 68020 CPU and 68881 floating-point unit implement the DP, while the 68851 functions as the memory management unit. Each node contains 16 Mbytes of memory.

The breadboard CP supports four bidirectional links, each of which consist of two unidirectional links running at 20 Mbits/s. The processor, Ethernet, memory boards, and CP connect to the VMEbus. The boards are commercially available; the CP is a custom design. Note that one crate houses five separate VMEbuses, one for each node. Only one node in each crate has an Ethernet board used for downloading code and inputting and outputting to the host computer.

The total main memory capacity of the Pooma prototype is 1.6 Gbytes. This large main memory encourages the use of



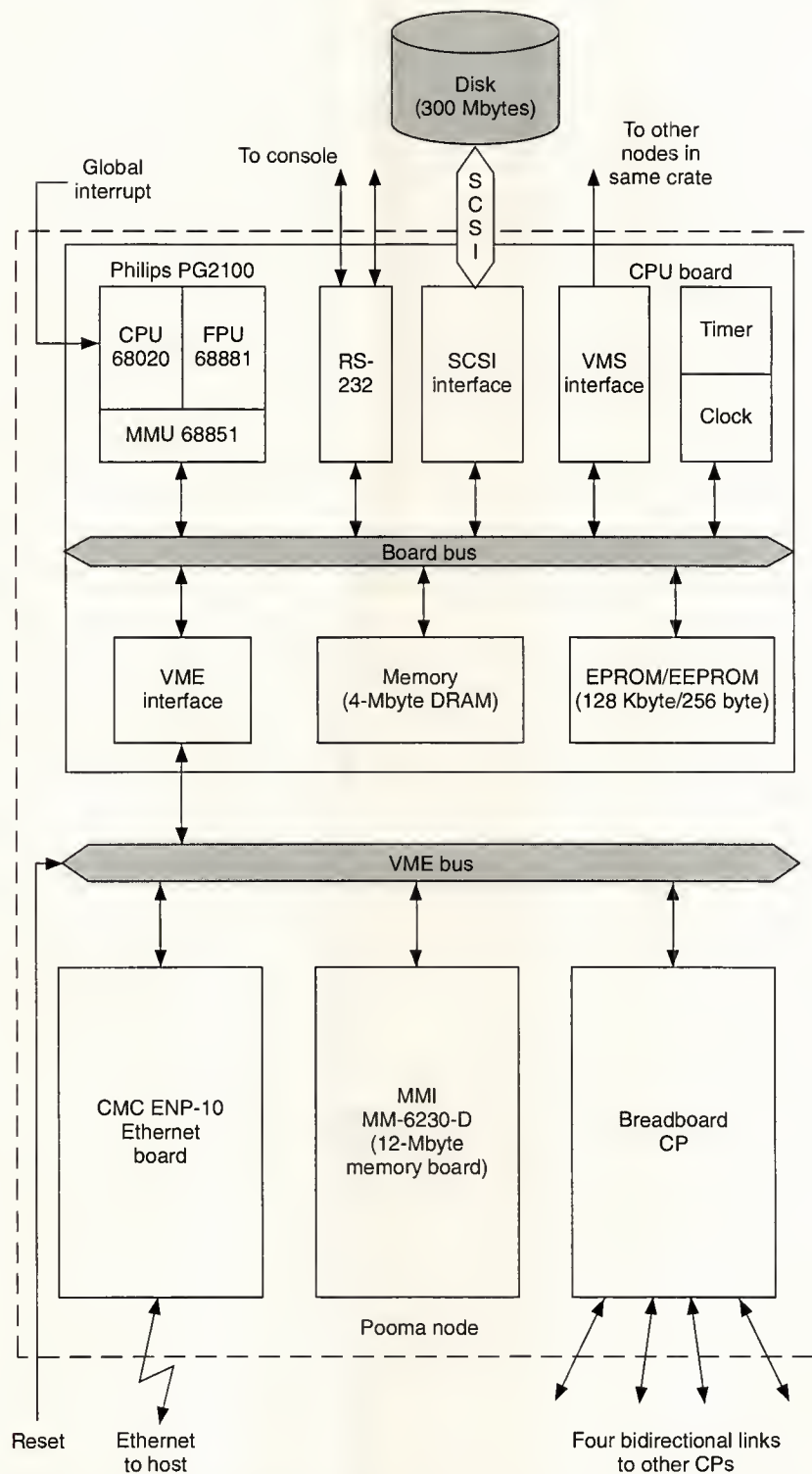


Figure 7. Pooma prototype node implementation.

special techniques (main memory database, inverted file method) to improve the performance of data-intensive applications.

The four node cabinets are grouped around the central switch cabinet. We added this switch unit to the prototype for experimental (and flexibility) reasons only; it is not a fundamental part of the Pooma decentralized memory architecture. The switch splits the 100-node machine into a number of smaller machines (arbitrary multiples of five nodes), so that more users can access the machine at the same time. This switch also allows experiments with different kinds of network topologies.<sup>20</sup> Another feature of the switch is that a faulty node or link can be repaired (or replaced) while the rest of the prototype remains operational. We simply configure the switch such that the faulty parts are not allocated to users of the machine.

The interface with the outside world occurs through one Ethernet cable to which the 20 Ethernet boards of Pooma connect. These boards run standard Ethernet software (TCP/IP). We've designed a software environment that allows users on the LAN to use the prototype from their own workstations. The software environment also controls the resources (nodes and switch) of the prototype and allocates clusters of nodes to users on request. Users specify the number of nodes and the kind of network topology. A request waits in a queue when the number of requested nodes is higher than the number of free nodes in the prototype. Every time the software environment allocates nodes and frees nodes, it automatically resets these nodes so the next user has a clean machine.

**Performance issues.** Van Beek, van Twist, and Vlot<sup>12</sup> describe experiments that show the Pooma communication network supports excellent node-to-node message passing, even in situations with very aggressive communication loads. Experiments included those on:

- *Communication load.* Included are the number of packets produced per second by each node in the system; burst versus no-burst traffic; and uniform versus hot-spot packet distribution. Packets within one burst transfer to the same destination and form a message together. These packets transfer into the CP of the sending DP at a very high speed. In the no-burst situation all messages contain just one packet. Uniform packet distribution means that any network node can become the destination of a packet with equal probability. Hot-spot packet distribution means that one node has a higher probability of being chosen as the destination of a packet than the other nodes.
- *Routing strategy.* Dynamic or static strategies exist.
- *Network topologies.* Torus, chordal ring, and extended chordal ring topologies are possible.<sup>12</sup>

The experiments measured packet delay and packet

throughput, both measures of network performance. We concluded the following from the experiments:

- *Throughput.* In high-network-load situations of up to 80 percent link occupation, the network does not incur much extra delay when compared to low traffic situations. An exception occurs in burst traffic situations when few alternative paths exist.
- *Dynamic routing.* In all cases dynamic routing diminishes network delay, and in the case of burst traffic it significantly increases the maximum throughput.
- *Topology.* When using static routing or when expecting little burst traffic, a topology with a low-average inter-node distance is most suitable. In burst traffic situations the use of more alternative shortest paths more than compensates for the increase in average distance.
- *Hot spots.* The network behaves extremely well under hot-spot loads. Until about 80 percent occupation on the highest loaded links, the average delay of a packet hardly increases when compared with the delay in low-load situations.
- *Scalability.* As long as the average link occupation remains below 80 percent of its maximum, only a small increase occurs in the delay as the nodes increase. There seems to be no reason that this network concept cannot be extended to several hundreds of nodes.
- *Comparison.* We found it difficult to compare the Pooma network with other machines for two reasons. The principles on which the machines are based differ, and no measurements exist for these machines to compare the situation for which the Pooma network was constructed—high throughput for many small messages. Compared to the iPSC/2,<sup>22,23</sup> the POOMA network is better suited for routing many smaller messages and for quickly varying communication patterns. The iPSC/2 is better suited for transferring large messages and for more static communication patterns.

## System software

The system software executing POOL programs on Pooma hardware consists of a compiler and a dedicated operating system. The compiler checks the POOL code and generates an executable program. The operating system offers an abstraction of the hardware toward the generated code and also takes over most distributed POOL tasks from the compiler. Furthermore, the operating system handles downloading and execution of the program and offers some facilities for performance analysis and debugging.

**The compiler.** The compilation of POOL programs consists of several steps. First the compiler checks the POOL code against syntactic and semantic errors. Next the compiler uses an intermediate language to compile the program into assembly code. The intermediate language describes a stack



machine, enhanced with the basic POOL primitives for communication, body execution, and object creation. After linking the compiled code of the several units together, the compiler offers the program to the parallel machine for execution.

The basic communication mechanism in POOL, which is used for almost every operation expressed in the language, is very difficult to implement efficiently in its full generality. Therefore we defined several kinds of objects that satisfy certain restrictions and allow a much cheaper implementation:

- *Server objects.* These objects do *not* run in parallel with the objects that send messages to them. They simply wait for a message, execute the corresponding method, send the result to the sender, and then wait for the next message. In other words a server object does not need a full-fledged process for itself. Instead, messages sent to them by an object on the same node can be processed by a "serve-yourself" mechanism: The sender executes the method code within its own process. In this way, we can replace the expensive message-passing mechanism with an inexpensive procedure call. We only need a semaphore to prevent several senders from executing methods in the same object simultaneously. (For processing messages sent from other nodes, special system processes execute the serve-yourself mechanism.)
- *Data objects.* These objects obey the same restrictions as server objects. In addition their methods are atomic, that is, they can execute from beginning to end without needing scheduling in between. Data objects do not need the semaphore that server objects use to control the access to their methods. In many cases, the procedure call is even replaced by in-line code. An example of a data object is a recordlike object with only put and get methods for accessing the variables. Several built-in classes such as arrays are also of this type.
- *Value objects.* In addition to satisfying the restrictions of data objects, value objects guarantee that their internal states never change after their creation. They are immutable. Programmers can replicate these objects without changing the semantics of a program. Whenever a reference to a value object is included as a parameter in an internode message, the runtime system copies the complete object. In this way, access to value objects is always guaranteed to be local and therefore inexpensive. Most built-in classes are of this kind, for example, integers, strings, tuples.

Only for the remaining kind of objects (we call them process objects) does the compiler generate full-fledged, but lightweight, processes that execute the objects' bodies. Since server, data, and value objects do not each need a process, many more objects than processes can exist on a node.

Since the performance gained by using specialized object implementations can be very large (see later), we decided that the compiler should not perform these optimizations unless the programmer is aware of them. Instead, the programmer indicates the intended object type by a *pragma*, a kind of comment to the compiler. The compiler then checks for confirmation that the restrictions are satisfied and only then performs the optimization.

**The operating system.** We specially constructed the operating system for execution of POOL programs. It roughly consists of three parts:

- The *nucleus*, which takes care of the resources offered by the Pooma hardware, that is, the memory, CPU, CP, and optional disk and Ethernet.
- The *POOL support*, which handles most of the distributed tasks related to POOL, most notably message handling, garbage collection, and object allocation.
- The *program support*, providing among other facilities downloading programs, a monitor on which to graphically display the execution, and profiling and debugging tools.

Because of the high rate of allocations and deallocations, we made the functionality of the memory manager as simple as possible. It only supports simple *alloc* and *free* primitives. Disk swapping is eliminated: Main memory stores all data.

We paid special attention to scheduling, because it occurs very frequently. We chose to support only lightweight processes. Furthermore, scheduling occurs only on request. As a consequence the compiler schedules as needed and tries to minimize the amount of data to be saved at those points. To prevent monopolization of the CPU, the compiler inserts some preemptive schedule points in loops, for example.

The operating system directly supports the basic POOL communication primitives. The message handler is distributed over the nucleus and the POOL support. The message handler of the nucleus provides a transport layer and supports three different message types. It supports a fixed-sized message for executing routines with a small number of parameters on a remote node; a copy message for copying data of any size to a predetermined location at the receiving node; and a variable-sized message. In the latter case, the receiving node in which the data is stored must allocate memory. The POOL support layer of the message handler offers two different buffering protocols, destination and global message.<sup>24</sup> Destination buffering assumes the message can be allocated at the receiver; it crashes the system when this is not the case. By contrast, global message buffering inserts the message in a distributed queue in this case and fetches the message later, when sufficient memory is available.

The programmer, considered primarily responsible for proper allocation of the objects, specifies allocation pragmas

Table 4. Timings for get access.

Object type	With runtime checks ( $\mu$ s)	Without runtime checks ( $\mu$ s)
Process	426.0	425.0
Server	2.8	2.2
Data	2.8	1.3
Value	2.0	1.3

Table 5. Timings for method access.

Object type	With runtime checks ( $\mu$ s)	Without runtime checks ( $\mu$ s)
Process	426.0	425.0
Server	7.0	6.5
Data	7.0	5.4
Value	6.2	5.4

in the POOL code. With such a pragma the programmer can specify a set of nodes on which a new object can be allocated. The operating system then attempts to allocate the object on one of these nodes.

A garbage collector removes redundant POOL objects (more specifically, objects that will no longer participate in the execution of the program). For this the operating system uses a distributed, on-the-fly, mark-and-sweep garbage collector.<sup>25</sup> The algorithm runs concurrently with the POOL program (on the fly) to exploit the available parallelism optimally. We used a mark-and-sweep strategy instead of reference counting because cyclic reference structures are very common in POOL.

**Performance.** Currently we are tuning the POOL implementation; therefore we have only a limited number of performance figures as yet.

Tables 4 and 5 display the effects of introducing server, data, and value objects. Table 4 lists the amount of time needed to call a method that simply extracts the value of an integer variable in a local object (a get operation). POOL provides a special notation for such a method, thus giving the compiler the opportunity to optimize the accesses with in-line code. When runtime checks are disabled, the compiler does not check to verify whether the object has been assigned a proper value nor perform deadlock checks. The table shows that local process communication should certainly be optimized further (we've paid little attention to that up to now). It also shows that for other object types the performance is very good (a comparable operation in a C program takes 1.0  $\mu$ s).

Table 6. Timings for memory management.

Operation	Small ( $\mu$ s)	Large ( $\mu$ s)
Nucleus alloc	20	67
Nucleus free	15	67
POOL alloc	25-30	77
POOL free	19	71

Table 7. Timings for process management.

Operation	Time ( $\mu$ s)
Switch	9
Nucleus switch	19
POOL switch	12
Nucleus insert into ready queue	14
POOL insert into ready queue	29-32

Table 5 lists results of a similar operation that enforces a full method call. Such an operation compares to a C routine call, which takes 4.7  $\mu$ s.

From these tables we can conclude that the POOL implementation can almost keep up with the C implementation, if the programmer makes use of the proper object types. We should also take into account that the POOL compiler does not use very advanced optimization techniques (such as interprocedural dataflow analysis), so some room for improvement still exists.

The memory manager employs the paging mechanism of the MMU provided by the DP board. Therefore a difference must be made between allocation requests smaller than the page size and larger than the page size. The timings of the memory manager depend on its usage. To evaluate the memory manager, we ran a typical POOL program (a parallel theorem prover) and measured the average allocation time. The results appear in Table 6. The first two operations involve timings at the nucleus level of the operating system, the latter two are measured at the compiler level. The POOL allocation time depends on the context in which it is called. These figures indicated that the memory manager is sufficiently fast to cope with most POOL applications.

We also measured the time it took for the process manager to switch among processes and insert processes into the ready queue. Table 7 lists the results. The bare switch does not save any context beyond the program counter. The other switch times reflect speeds for nucleus-level processes and



POOL processes. A similar distinction is made for insertion in the queue, in which the POOL timing depends on the context where the operation is performed. We can conclude that the process manager is sufficiently fast even for programs with a relatively small grain size.

## Applications

Probably one of the most important leaps forward made in the Pooma project is the easy access for application programmers to the full power of a parallel computer system. Many former parallel computer projects concentrated on the hardware aspects, resulting in systems that were fast but hard to program. The Pooma project radically broke with this tradition. A main goal was to build a system on which complex applications could be programmed easily and run efficiently. Therefore we did not choose the standard applications for parallel computer systems, which are easy to program on all systems. Instead, we concentrated especially on those problems that are difficult to map to traditional parallel machines—symbolic applications. These applications in general have irregular computation patterns, which often depend on the input data. Since we did not (and still do not) believe automatic techniques for extracting parallelism from programs are sophisticated enough to achieve an efficiency comparable to programs with explicit parallelism, we chose the latter alternative.

We found one of the most demanding applications, as far as programmability is concerned, to be the Horn Clause theorem prover,<sup>26</sup> which can also be used to execute logic programs similar to Prolog. This theorem prover makes use of a breadth-first strategy and uses Or-parallelism. The prover provides a mechanism to prohibit the repeated examination of the same goal. Further, the prover makes use of counter examples to prune the search tree. In this application new goals derive from old ones and axioms all the time, the connection structures between these goals are subject to changes, and many goals become obsolete after some time. For such an application the POOL features for creating new objects dynamically and changing their interconnection patterns seem indispensable. Furthermore, the garbage collector takes care of clearing away the obsolete objects.

All kinds of processes manipulate the goals in the Horn Clause prover and are transferred from one to the other. Here, the transparency of the process locations and the network is very important. Objects or references to them can be sent from one process to the other without the sender having knowledge of the underlying network structure, or even knowledge of the location of the receiver or the communicated object.

The practical work on parallel programs gave rise to a better understanding of the incorporation of parallelism in applications. We observed that processes of different types

typically occur only at the highest design levels. These processes correspond to different tasks in the application. They do not give rise to much parallelism but are important for the modular structure of the program. In general the communication patterns at this level are irregular. Most designers usually introduce parallelism at the lower design levels, in which several identical processes occur that perform the same task on different data items. At this level we recognize regular communication structures and simple communication protocols. We used certain basic communication structures in a number of applications:

- *Pipelines.* We used pipelines when a number of transformations had to be applied to a large set of similar data items.
- *Trees.* We typically used these structures for problems that can be solved by divide-and-conquer techniques.
- *Blackboards.* In blackboards many processes communicate via a shared set of data. They take data items from the set, process them, and enter new items to it.
- *Hypercubes.* These structures can serve to broadcast messages to or to collect data from a number of processes.
- *Shuffle exchanges.* The most important use here is in sorting and in processing fast Fourier transforms.

As far as the efficiency of programs is concerned, we found that discovering sources for parallelism was seldom a problem. The difficulty mostly was to exploit this parallelism in a sensible way. The main causes for inefficiency are processor idle time and overhead introduced by communication. Often the reduction of either of those causes results in an increase of the other, making it essential to find a good balance. Bosco, Cecchi, and Moiso<sup>11</sup> offer the best example of this relationship. Here, a dynamic load-balancing strategy, introducing as little communication as possible, decreases idle time. All aspects of efficiency are very sensitive to which processes are grouped on the same node. Therefore the programmer's ability to influence the allocation of objects to nodes by adding pragmas to the program is very important.

ESPRIT PROJECT 415 COVERED THE INVESTIGATION of six different language and execution models for parallel symbolic computer systems and design of corresponding languages and system implementations. The project resulted in a major contribution to the theoretical and practical understanding of this novel type of system and to the emergence of a scientific community in this area in Europe.

The single-project framework provided a forum for discussions and comparisons among the different ways to achieve parallel symbolic systems. In addition, the efforts on the object-oriented style delivered a scalable, highly parallel

system with a novel, high-performance communication architecture. The parallel object-oriented language provides programmers with a very flexible and dynamic model of parallelism. It permits concentration on the essential aspects of a parallel design, while abstracting from many details of the underlying system. Despite the dynamic model, the implementation of the language results in excellent performance. ■

## References

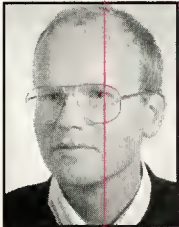
1. J.W. de Bakker, W.P. deRoever, G. Rozenberg, eds., "Current Trends in Concurrency," *Proc. LPC/ESPRIT Advanced School in Springer Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Heidelberg, Vol. 224, 1986.
2. P.C. Treleaven and M. Vanneschi, eds., "Future Parallel Computers," *Proc. ESPRIT Summer School in Springer LNCS*, Vol. 272, 1987.
3. J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, eds., *Proc. Parle: Parallel Architectures and Languages Europe, Vol. 1: Parallel Architectures, Vol. 2: Parallel Languages*, in *Springer LNCS*, Vols. 258 and 259.
4. E.A.M. Odijk, M. Rem, and J.-C. Syre, eds., *Proc. Parle: Parallel Architectures and Languages Europe*, Vols. 1 and 2 in *Springer LNCS*, Vols. 365 and 366.
5. P.H.M. America and M. Beemster, "A Portable Implementation of the Language POOL," *Proc. TOOLS: Technology of Object-Oriented Languages and Systems*, pp. 347-353.
6. A. Augusteijn, "The Elegant Compiler Generator System," *Proc. Int'l Workshop on Attribute Grammars and Their Applications in Springer LNCS*, Vol. 461, Sept. 1990, pp. 238-254.
7. E. Aposporidis and F. Lohner, "Parallel Multi-Level VLSI Simulator: An Object-Oriented Approach," *Proc. European Simulation Multiconference*, S. Tucci, Rome, pp. 361-366.
8. G.L. Burn, "Developing a Distributed Memory Architecture for Parallel Graph Reduction," *Proc. Conpar 88*, Cambridge University Press, UK.
9. B. Bergsten et al., "An Advanced Database Accelerator," *IEEE Micro*, Vol. 8, No. 5, Oct. 1988, pp. 47-63.
10. J. Rohmer, R. Lescœur, and J.M. Kerisit, "The Alexander Method: A Technique for the Processing of Recursive Axioms in Deductive Databases," *New Generation Computing*, Vol. 4, 1986.
11. P.G. Bosco, C. Cecchi, and C. Moiso, "An Extension of WAM for K-LEAF: A WAM-Based Compilation of Conditional Narrowing," *Proc. Sixth Conf. Logic Programming*, MIT Press, Cambridge, Mass., 1989.
12. W.J. van Beek, R.A.H. van Twist, and M.C. Vlot, "Evaluation of the Communication Network of Pooma," *Proc. Int'l Conf. Parallel Processing*, Vol. 1, 1990, pp. 498-507.
13. P.H.M. America, "Issues in the Design of a Parallel Object-Oriented Language," *Formal Aspects of Computing*, Vol. 1, No. 4, 1989, pp. 366-411.
14. J.J. Aalbersberg and F.W. Sijstermans, "InfoGuide: A Full-Text Document Retrieval System," *Proc. Dexa: Database and Expert Systems Applications*, Springer, 1990, pp. 12-21.
15. P.M.G. Apers, M.L. Kersten, and A.C.M. Oerlemans, "Prisma Database Machine: A Distributed, Main-Memory Approach," *Proc. Int'l Conf. Extending Database Technology*, 1989.
16. J.M. Jansen and F.W. Sijstermans, "Parallel Branch-and-Bound Algorithms," *Future Generation Computer Systems*, Vol. 4, North-Holland, Amsterdam, 1989, pp. 271-279.
17. W. Bronnenberg et al., "DOOM: A Decentralized Object-Oriented Machine," *IEEE Micro*, Vol. 7, No. 5, Oct. 1987, pp. 52-69.
18. P.C. Treleaven, ed., *Parallel Computers: Object-Oriented, Functional and Logic*, John Wiley and Sons, Inc., New York, 1990.
19. J.K. Annot and P.A.M. den Haan, "POOL and DOOM: The Object-Oriented Approach" in *Parallel Computers: Object-Oriented, Functional and Logic*, John Wiley and Sons, Inc., 1990, pp. 47-79.
20. R.A.H. van Twist and E.A.M. Odijk, "Networks for Parallel Computers" *Proc. VLSI and Computers, Compeuro*, 1987, pp. 779-782.
21. J.K. Annot, "A Deadlock-Free and Starvation-Free Network of Packet Switching Communication Processors," *Parallel Computing*, Vol. 9, 1989, pp. 147-162.
22. R. Arlauskas, "iPSC/2 System: A Second Generation Hypercube," *Proc. Third Conf. Hypercube Concurrent Computers and Applications*, 1988, pp. 33-36.
23. T.H. Dunigan, "Performance of a Second Generation Hypercube," tech. report ORNL TM-10881, Oak Ridge National Laboratory, Oak Ridge, Tenn., 1988.
24. P.A.M. den Haan and Frans Hopmans, "Efficient Message Passing in Parallel Systems with Limited Memory," *Proc. Conpar 88*, Cambridge University Press.
25. A. Augusteijn, "Garbage Collection in a Distributed Environment," *Proc. Parle in Springer LNCS*, Vol. 259, pp. 75-93.
26. J.M. Jansen and F.W. Sijstermans, "Implementation of a Parallel Horn Clause Theorem Prover, ESPRIT Project 415-A," Doc. No. 371, Philips Research Laboratories Eindhoven, The Netherlands, 1988.



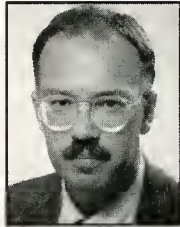
**Pierre H.M. America** holds responsibility for the design of the POOL parallel object-oriented language at Philips Research Laboratories. He works on formal semantics and verification techniques for these languages.

America holds doctorates in mathematics and computer science from the University of Utrecht and the Free University of Amsterdam, The Netherlands. He is a member of the Institute of Electrical and Electronic Engineers, the Association of Computing Machinery, and EATCS (the European Association of Theoretical Computer Science).

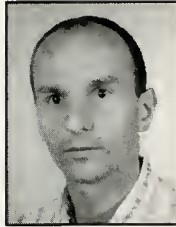




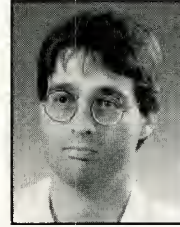
*Hulshof*



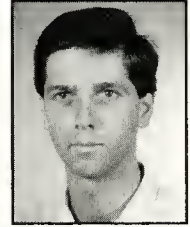
*Odijk*



*Sijstermans*



*van Twist*



*Wester*

**Ben J.A. Hulshof** is project leader of the parallel processing activities within the Computer Architecture Department. He has worked on the Dutch Prisma project since 1986. His technical interests include parallel programming systems, garbage collection, and special-purpose distributed operating systems.

Hulshof received his bachelor's and master's degrees from the Technical University of Twente, The Netherlands.

**Eddy A.M. Odijk** heads the Computer Architecture Department at Philips Research Laboratories and serves as project leader of ESPRIT project 415. Previously, he worked on VLSI designs for communication protocols and DSPs. His professional interests include parallel and distributed architectures and multimedia systems.

Odijk received his master's degree in electronics from the Technical University of Twente. He is a member of the IEEE and the ACM.

**Frans Sijstermans**, a member of the Philips ESPRIT 415 project team, works on applications, specializing in the design of parallel programs. His technical interests include applications of parallel computers in symbolic computing, for example, in artificial intelligence, information retrieval, and image processing.

Sijstermans received his master's degree in computer science from the Technical University in Eindhoven, The Netherlands.

**Rob A.H. van Twist** works as a system engineer in the Computer Architecture Department of Philips. He previously worked on a VLSI digital signal processor. His current interests include parallel computer systems, processor architectures, and network architectures for highly parallel computer systems.

Van Twist holds a bachelor's degree from the Technical High School of Vlissingen, The Netherlands.

**Rogier H.H. Wester** is a member of the Computer Science Department at Philips. In the past he worked on a database for an automobile information and navigation system. Currently his technical interests include parallel computers, operating systems, and parallel applications.

Wester received his master's degree in mathematics and computer science from the University of Eindhoven.

Address questions concerning this article to Eddy Odijk, Philips Research Laboratories, PO Box 80.000, 5600 JA Eindhoven, The Netherlands; or e-mail at [odijk@prles6.prl.philips.nl](mailto:odijk@prles6.prl.philips.nl).

---

### Reader Interest Survey

Indicate your interest in this article by circling the appropriate numbers on the Reader Service Card.

Low 153

Medium 154

High 155

---

## Transputers

*continued from p. 19*

supporting windowing systems enjoy increasing popularity. In both cases, the workstation formerly carried out the corresponding processing. Similarly, one expects techniques for high-performance systems developed on machines such as Supernode to migrate into next-generation embedded systems.

### Transputer architecture

The development of transputer architecture involved four main objectives:

- it created a commercial product range that set new standards in ease of programming and engineering;
- it provided maximum performance to the user;
- it allowed the exploitation of future developments in VLSI technology within a compatible family; and
- it created a programmable component for building systems with large numbers of concurrent computing components.

A transputer contains a processor, memory, and a number of standard point-to-point communications links—all integrated into one silicon chip (as shown in Figure 1). An external memory interface extends the on-chip memory. When appropriate, transputers also incorporate special-purpose processing and/or interfacing capabilities. Separating the external memory interface (for local memory) from the communications optimizes performance and minimizes contention.

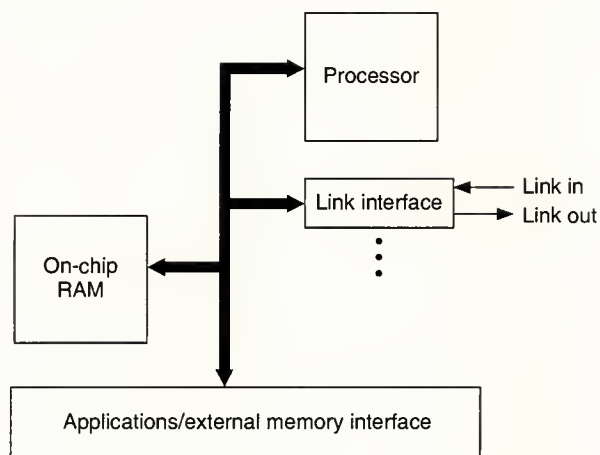


Figure 1. Processing and interfaces in transputer architecture.

A system is constructed from one or more transputers operating concurrently and communicating through standard links. The programming language Occam (see box) formalizes the computational model. Occam describes a system as a collection of processes and communications that operate concurrently and communicate through channels.

### Transputer processing

The transputer directly implements the Occam model of concurrency. A hardware scheduler allows any number of Occam processes to share a single processor, and transputer instructions implement Occam message passing. An application designer can configure a collection of processes ready for execution on a network of transputers. Each transputer executes a component process and transputer links implement Occam channels.

Both internal and external communications use the same instructions, allowing for Occam program reconfiguration (such as using a different processes-to-processors allocation) without recompilation. In particular, an application designer can configure an Occam program to execute on a small number of transputers for low cost or on a larger number of transputers for high performance.

The transputer processor supports fast interrupt response by providing two levels of priority. (Typically, the interrupt response is less than 1 microsecond on a 20-MHz clock transputer; worst case, it is less than 4 microseconds). Using the ALT construct (a key word in Occam meaning alternative), a high priority process waits for the first of several inputs to become ready. It then executes the specific piece of code to respond to the particular interrupt.

The processor treats access to a timer as an input. In a delayed input, the process waits until the timer reaches an appropriate value. The processor supports an arbitrary number of timer inputs. A programmer can also use a timer input within an ALT construct as a time-out on a communication.

Developing the transputer instruction set involved five design objectives:

- to implement Occam effectively, so that high-level language usage results in the effective use of silicon capability, and that highly concurrent programs execute with minimum overheads;
- to implement Occam simply and directly to facilitate easy, straightforward program compilation, and to ensure that lower level programming is unnecessary;
- to provide word-length independence, so that a program executes using processors of different word lengths without recompilation;
- to provide position independence, so that programs and workspaces are allocated anywhere in memory after recompilation; and

*continued on p. 78*



## Basic Occam concepts

The Occam language<sup>3</sup> programs concurrent, distributed systems. The word *distributed* emphasizes the unsuitability of previous languages in this area. Occam describes a system as a collection of concurrent processes that communicate with each other and with peripheral devices through channels. Concurrent processes do not communicate via shared variables; thus Occam is particularly suitable for programming systems with no memory sharing between processors.

Occam provides three primitive processes:

$v := e$     Assign expression  $e$  to variable  $v$   
 $c! e$     Output expression  $e$  to channel  $c$   
 $c? v$     Input from channel  $c$  to variable  $v$

Occam provides constructs that combine primitive processes:

SEQ	Components execute one after another (sequential)
PAR	Components execute together (parallel)
ALT	First ready component executes (alternative)

The language also provides IF and WHILE constructs.

A construct is itself a process and it may be used as a component of another construct. Occam syntax uses indentation to indicate program structure.

A programmer writes parallel programs by using channels, inputs, and outputs combined in parallel and alternative constructs. Each Occam channel provides a communication path between two processes. Communication is synchronized and takes place when both the inputting and the outputting processes are ready. Data to be communicated is copied from the outputting process to the inputting process, and both processes continue.

An ALT process waits for input from any one of a number of channels. ALT takes input from the first to be used for output by another process.

Occam provides a replicated constructor. For example

```
SEQ i = base FOR count
  a[i] := i
```

implements as a loop and is equivalent to

```
SEQ
  a[base]           := base
  a[base + 1]       := base + 1
  .....
  a[base + count - 1] := base + count - 1
```

Replication used with PAR provides arrays of similar processes. ALT and IF can also be replicated.

Using a construct as a component in another construct makes it possible to design a system as a set of nested processes (for example, by using PAR within PAR as shown in Figure C). The messages input and output on the channels of a process fully specify the process, and this completely hides its internal structure from the outside world.

Internally, the programmer can structure the process itself as a set of nested processes. At any level of design, the designer works with a small and manageable set of processes.

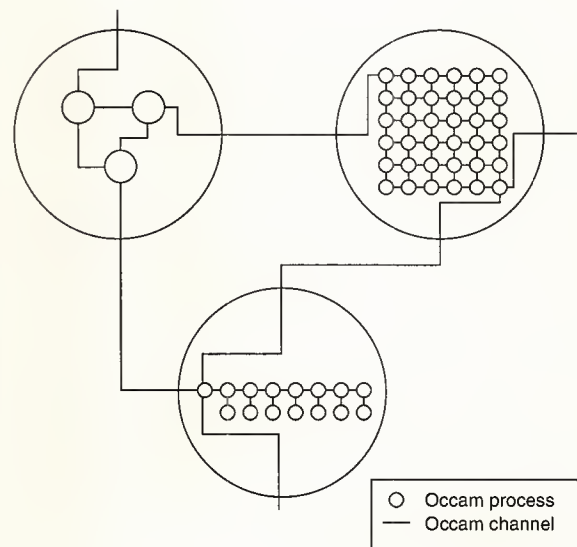


Figure C. Design of nested Occam language processes.

- to provide low-latency response to communications with external devices.

The resulting design<sup>1</sup> uses a simple linear address space, six functional registers for sequential programming (see Figure 2), and additional registers as queue pointers to support concurrency.

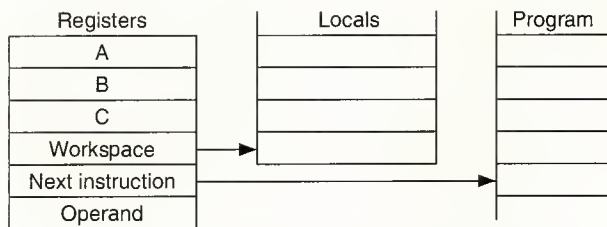


Figure 2. Function of transputer registers for sequential programming.

The six registers for sequential programming are the

- workspace pointer that points to a storage area containing local variables,
- instruction pointer that points to the next instruction to be executed,
- operand register used to form instruction operands, and
- A, B, and C registers that form an evaluation stack. This stack holds the operands and intermediate results for expression evaluation.

The hardware scheduler allows for the combined execution of any number of processes through the sharing of processor time. At any time, a concurrent process is active (either currently executing or on a list awaiting execution) or inactive (either ready to input, ready to output, or waiting until a specified time).

A list holds active processes awaiting execution. This linked list of process workspaces uses two registers in implementation—one that points to the first process on the list and one that points to the last process. The hardware scheduler maintains two such lists—one for high-priority processes, the other for low-priority processes.

The implementation of the instruction set uses a single level of microcode. Many instructions execute in one cycle (50 ns on a 20-MHz transputer); many of the rest execute in two cycles. Some complex functions (such as block move) take an arbitrary number of cycles. These instructions still provide higher performance than possible with software.

To limit the latency figure for switching between low and high priority, time-consuming instructions allow a switch during execution. Consequently, the processor never takes

more than 4 microseconds to switch between low priority and high priority.

A context switch between processes executing at low priority occurs only when the evaluation stack contains no useful contents. With minimal need to save and restore registers, the processor implements concurrency very efficiently.

The instruction format uses very compact encoding based on 1-byte instructions. Prefixing instructions are used to form long operands. The instruction size is independent of the word length. In general, a program requires much less storage to hold it than an equivalent program in a conventional or RISC microprocessor. Since a program requires less storage to represent it, fetching instructions use less memory bandwidth. As the transputer accesses memory one word at a time, the processor receives several instructions for every fetch (depending upon the number of bytes in a word).

In addition to Occam, high-performance compilers for C, Fortran, Pascal, and Ada have been implemented for the transputer.

### Transputer communications

A link between two transputers implements a pair of Occam channels, one in each direction. Two one-directional signal lines connect a link interface on one transputer to a link interface on the other transputer. Each signal line carries data and control information.

Communication through a link involves a simple protocol, which supports the synchronized communication of Occam. The protocol provides for the transmission of an arbitrary sequence of bytes, which allows transputers of different word lengths to communicate.

Each byte transmits as a start bit, followed by a one bit, 8 data bits, and a stop bit (see Figure 3a). After transmitting a data byte, the sender waits until receiving an acknowledgment, which consists of a start bit followed by a zero bit (see Figure 3b). The acknowledgment signifies both that a process received the acknowledged byte, and that the receiving link is ready to receive another byte. The sending process proceeds only after receiving acknowledgment for the final byte.

Data bytes and acknowledgments multiplex down each signal line. An acknowledgment transmits as soon as recep-

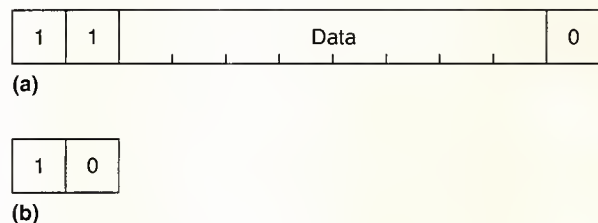


Figure 3. Formats of data links (a) and acknowledgment (b).



tion of a data byte begins (provided a process is waiting for it and room to buffer another data byte is available). Consequently, transmission may continue without delays between data bytes.

As the transputer uses transistor-transistor-logic-compatible (TTL) signals, the applications engineer can extend the links by inserting industry-standard line drivers and receivers.

The design of the links makes engineering of transputer systems easy. Irrespective of internal performance, all transputers use a 5-MHz clock for frequency reference. The low frequency simplifies the clock distribution in large transputer systems. The communications system does not require a phase reference. Therefore, it is not necessary for all transputers to operate on the same clock. The flexibility to use a number of clocks enables interworking between independently designed (sub)systems.

The use of point-to-point serial communications, instead of buses, offers the following advantages:

- simplified board layout and backplane design;
- increased communications bandwidth, as many links in a system operate concurrently; and
- easy interconnection of devices with different word lengths and performance.

Transputers with different word lengths and performance all interwork together, ensuring the easy upgrading of systems as the technology advances. It is not necessary to downgrade a connected set of components to the performance of the slowest component!

Each transputer contains a separate communications engine, allowing communications to proceed in parallel with the execution of processor instructions. Indeed, many applications completely overlap communications and processing, maximizing overall system throughput.

Two link adapters and a link switch add to the flexibility and use of communications. The link adapters provide an interface between a link and a byte-wide port. The Inmos C004 link switch provides a crossbar switch between 32 links, controlled by a separate configuration link. The C004 is cascable, allowing for the construction of arbitrary networks of transputers (limited only by the number of links on each transputer; most transputers contain four links).

## Programming paradigms

While designing transputer-based hardware to perform at any desired level of performance is easy, one must ensure that the software configuration exploits the hardware architecture. Software structured as a sequential program with conventional compilation operates no faster on 10 transputers than on one!

For embedded systems, the design of software architecture and hardware architecture optimally occurs hand-in-hand,

resulting in essentially a system design activity.<sup>5</sup>

To configure a program for a network of transputers, the applications designer identifies the parallelism. The designer subsequently maps the parallel processes onto the transputer network to optimize the system according to the design criteria (such as maximized performance and minimized latency).

Experience gained to date with parallel systems—particularly with ESPRIT projects—identified a number of programming paradigms that help to structure systems designs.<sup>6</sup> These paradigms can be written in languages such as Ada,<sup>7</sup> or can employ parallel extensions or libraries in C or Fortran. However, Occam<sup>3</sup> describes these paradigms most conveniently.

Descriptions of the paradigms for algorithmic parallelism, geometric parallelism, and farming appear below:

- 1) *Algorithmic parallelism.* The designer splits the application into functional units. In simple cases, these units can form a pipeline; in general, they can form more complex structures such as feedback loops. The various stages potentially operate in parallel. With a modest amount of buffering, the communication of data or partially computed results between stages will, in many cases, completely overlap processing (eliminating communications overhead). The structure maps onto one or more transputers, up to the number of components in the structure, with the communications performed internally or via transputer links, as appropriate. For a pipeline structure, its slowest stage limits the maximum performance.
- 2) *Geometric parallelism.* Designers partition the design on a regular basis. For example, they can divide a screen image into quadrants, or a matrix operation into submatrices. A separate process performs the computation for each partition. The processes often operate independently or interact only with immediate neighbors. A particularly beneficial use of geometric parallelism involves scaling up the size of a problem to be solved (such as performing weather forecasting on a finer mesh).

Designers usually implement geometric parallelism by allocating processes straightforwardly onto a physical regular network. Geometric parallelism usually attains maximum performance by considering granularity issues. The processes communicate very efficiently with their immediate neighbors when executing on the same processor. The processes communicate less efficiently with immediate neighbors when executing on separate processors. Where one process is allocated to each processor (see Figure 4a on the next page), communication costs dominate performance in most instances. With all processes on a single processor, computation time dominates performance.

Computation and communication achieve balance at an intermediate level of granularity (see Figure 4b). This

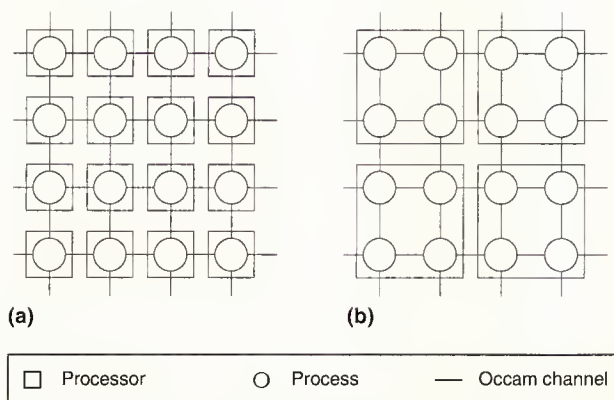


Figure 4. A simple allocation of geometric parallelism granularity (a); a balanced allocation of geometric parallelism (b).

balance results from a boundary-to-area effect (for two-dimensional grids) in which the amount of communication at the boundary varies linearly with the grain size, and the area of computation varies as the square of the grain size. In comparing the two figures, the boundary-to-area effect in Figure 4b results in four times the amount of computation, but only twice the communication as found in Figure 4a. A similar surface-to-volume effect occurs for three-dimensional grids.

- 3) *Farming*. Designers divide the application into small, similar pieces (for example, when needed to process a large number of similar data items). Each transputer in a network provides a server process, and a master process allocates (or "farms out") work to the servers as they become free. Farming provides two major benefits. First, it automatically balances the load—a server completing one piece of work immediately proceeds to the next. Second, it functions relatively independent of the topology—any reasonably linked network works well. The limit to performance is the rate of dispensing work and handling results.

The Occam model of parallelism offers a significant benefit: Messages sent and received completely define a process. The designer can structure a process internally as a set of processes, thereby using any desired level of nesting. A very powerful technique, therefore, combines the above paradigms in one application, such as a farm of geometric-array servers functioning with pipeline components. (The earlier Occam box diagrams the Occam program that encapsulates all three paradigms within a simple top-level structure.)

The designer can use these models to easily structure applications to define a large amount of parallelism. The nar-

row and easily defined interface specifications allow for easy reasoning about an application's correctness.

### The next-generation transputer

For the past two years, a team at Inmos's Bristol, England, design center has been working to enhance the transputer's performance and suitability for embedded systems. From this work, a new product family—based around a new processor code-named H1—will be launched in Spring 1991.<sup>8</sup>

The team's design goal called for establishing a new standard in single-processor performance while enhancing the transputer family's position as the premier multiprocessing microprocessor. It also required maintaining upward compatibility with existing transputer products.

To meet these goals, the team developed a new microarchitecture that implements the same instruction set as the existing Inmos T805 transputer. The H1 provides an order-of-magnitude increase in performance, combined with enhanced capabilities to support the software standards emerging in the embedded systems marketplace.

The H1 architecture includes such key features as a pipelined, superscalar processor combined with on-chip cache RAM, and improved communications that provide a new degree of freedom in multiprocessor programming.

To complement the H1 transputer, Inmos is now designing a range of network communication products based on a new 100-Mbit/s link protocol. The protocol supports the dynamic routing of messages between processors.

### H1 performance

The H1 provides a peak performance in excess of 150 MIPS (million instructions per second) and 20 Mflops (million floating-point operations per second) and a sustained performance exceeding 60 MIPS and 10 Mflops. It maintains instruction-set compatibility with the T805.

A number of design features contribute to the achievement of these performance levels. The processor itself uses a pipelined, superscalar architecture, which executes up to eight instructions on each clock cycle and operates at a clock speed of 50 MHz. The number of cycles required to execute many instructions—such as integer and floating-point multiply, and logical shift—decreases significantly.

Unlike other superscalar machines, the H1 architecture does not require an advanced compiler to schedule the different functional units in the processor. Hardware controls the flow of multiple instructions through the pipeline. It is not necessary to modify existing compilers or recompile source code.

An advanced submicron, CMOS (complementary metal-oxide semiconductor) process allows a high transistor count and high clock frequency operations. This process enables the implementation of a sophisticated processor and provides 16 Kbytes of on-chip cache memory.

The move to a cached architecture is a radical develop-



ment. The 16-Kbyte cache is sufficiently large to achieve high hit rates for most applications. The H1 still allows for directly addressed, on-chip RAM for applications containing only small amounts of memory, or ones intolerant of indeterminate performance caused by cache-line misses.

The design team took great care to ensure that the H1 transputer will provide high performance levels in low-component-count systems. For example, the H1 will provide a programmable memory interface with a 64-bit data bus sustaining high data transfer rates for cache-line refill. The interface supports four independent banks of external memory, and the timing for each bank is configured independently from software. For example, an application designer can choose to fill two banks with dynamic RAM—one bank with virtual RAM, the other with peripherals. Such a system frequently requires no external support logic.

### **H1 error-handling and user-mode processes**

The H1 transputer hardware supports the same scheduling algorithms used by current-generation transputers, such as the T805. In addition, on the H1 transputer each process may use a second process (known as a trap handler). When an error—such as an integer overflow or a floating-point error—occurs, control transfers to the trap handler. The trap handler copes with the error in software in all cases before it (in most instances) returns control to the process in which the error occurred.

The H1 also supports a separate user mode. This mode prevents privileged instructions (including communications and scheduling instructions) from executing. It checks and translates all memory accesses from a logical to the physical address space.

---

## ***H1 transputer enhancements allow programmers to write more efficient real-time kernels.***

---

Memory protection and address-translation mechanisms specifically support secure programming and debugging in embedded systems. For dedicated (single-user) systems, the protection aids the detection of programming errors. For multiuser, general-purpose computing systems, it protects users and the operating system from erroneous programs.

The protection and translation mechanisms are optimized for the requirements of embedded systems. These mechanisms allow the processor to execute code in protected (user) mode at the same speed as normal processes without the performance overhead involved in supporting page-based, virtual memory.

In contrast, but in keeping with its intended market, additional H1 transputer enhancements allow programmers to write more efficient real-time kernels. These enhancements access and control the state of the machine, the process and timer queues, and time slicing and interruptability mechanisms.

### **New freedom**

A limitation in exploiting existing transputer networks is the need to match the parallel structure of the algorithms used to the interconnectivity provided. In the worst case, a specific machine possesses a fixed topology. In the best case (a totally reconfigurable architecture such as the Supernode), the limitation of four links per transputer restricts mapping. This limitation can result in poor software portability, nonoptimum design, and scalability problems.

The H1 product family largely eliminates this problem by providing hardware to allow transputer connections via a low-latency communication network. It supports communication channels between any two processes anywhere in the network.

The hardware simplifies programming because designers do not need to consider how to allocate processes to the transputer network until after completion of the program writing. They can use different allocations on different machines and they can change the allocation to optimize performance. In addition, it is possible—at least in principle—to let the compiler make this allocation, effectively removing all configuration details from the program. Pountain<sup>9</sup> discusses the communications capabilities of the H1 product family in greater detail. The following paragraphs sum up these capabilities.

The H1 transputer itself contains a separate communications processor, which multiplexes a large number of logical communication links (virtual links) along each of its physical links. Each virtual link supports one Occam channel in each direction.

The communications processor transmits messages as a sequence of packets, which all contain 32 bytes of data except the last packet. Each message packet starts with a header, which routes the packet through the communication network and identifies the destination virtual link on the remote transputer.

Designers construct a separate communications network using the Inmos C104 routing device and can use one C104 to connect small numbers of H1 transputers. In larger systems, designers can use C104 connections to form a hypercube, a multidimensional grid, or a tree network.

Each C104 provides 32 bidirectional links. The header of each packet arriving on a link input determines the link on which to output the packet. As soon as the link output is free, the whole packet transmits through it.

An algorithm known as interval labeling decides through which link to send a packet. In interval labeling, each output

link is associated with a continuous set of header values (an interval). The header of an incoming packet lies within only one range, and the packet transmits to the associated link. Optimum, deadlock-free labeling schemes exist for each of the common network topologies.

The C104 provides additional facilities to connect networks together and reduce the impact of message congestion on worst-case latency and bandwidth in heavily loaded networks.

THE TRANSPUTER IS WELL ESTABLISHED as a highly cost-effective processor, particularly in embedded applications. It provides unique advantages for applications that require more than one processor, and serves as the basis for many research programs in parallel computing.

Inmos developed the current generation of transputers as general-purpose components for special-purpose machines. The introduction of a higher performance transputer—supporting virtual communications, memory protection, and other advances—represents a significant step toward the development of general-purpose, multiprocessing computing systems. Research continues on the architecture to effectively exploit the full capabilities of VLSI technology during the 1990s. Meanwhile, the H1 provides highly efficient implementations of conventional operating systems and real-time kernels. It greatly reduces the cost of porting existing software and upgrading existing applications to take advantage of the transputer's capabilities. ■



**Colin Whitby-Stevens** manages the Central Technology Group in the Microprocessor Design Division of Inmos Limited. His interests include microprocessor architecture, parallel programming and system design, and programming languages.

He recently chaired a series of Commission of European Countries workshops—involving more than 70 companies and organizations—to establish the Open Microsystems Initiative. Prior to joining Inmos, he was a lecturer in computer science at the University of Warwick, where he developed a multiprocessor operating system for the Modular One computer. He subsequently established the Warwick Distributed Computing Research Project, which developed some of the key concepts subsequently exploited in the Inmos transputer architecture.

Whitby-Stevens holds a BSc degree in mathematics from Hull University, and the Diploma in computer science and PhD from Cambridge University.

He authored 30 technical papers and coauthored *BCPL—The Language and Its Compiler*. He is an affiliate of the IEEE Computer Society, a member of the Association of Computing Machinery, and an associate member of the British Computer Society.

Address questions concerning this article to Colin Whitby-Stevens, Inmos Limited, 1000, Aztec West, Almondsbury, Bristol, BS12 4SQ, United Kingdom.

## References

1. A. Baker, "A Signal Achievement," *Parallelogram International*, Vol. 2, No. 29, Aug. 1990, pp. 10-11.
2. M. Homewood et al., "The IMS T800 Transputer," *IEEE Micro*, Vol. 7, No. 5, Oct. 1987, pp. 10-26.
3. Inmos Limited, *Occam 2 Reference Manual*, Prentice-Hall, London, 1988.
4. Inmos Limited, *Transputer Instruction Set—A Compiler Writer's Guide*, Prentice-Hall, 1988.
5. Inmos Limited, *Communicating Process Architecture*, Prentice-Hall, 1988.
6. A.J.G. Hey, D.J. Pritchard, and C. Whitby-Stevens, "Multiparadigm Parallel Programming," *Proc. Hawaii Int'l Conf. System Sciences*, IEEE, New York, 1989, pp. 716-725.
7. J. Barnes and C. Whitby-Stevens, "High Performance Ada Using Transputers," *Defense Computing*, Vol. 1, No. 5, Sept./Oct. 1988, pp. 45-49.
8. C. Dyson, "Inmos H1 Architecture Revealed," *New Electronics*, Vol. 23, No. 8, Sept. 1990, pp. 21-24.
9. D. Pountain, "Virtual Channels: The Next Generation of Transputers," *Byte* (European and World Edition), Vol. 15, No. 4, Apr. 1990, pp. 3-12.

## Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156

Medium 157

High 158



## EDS

*continued from p. 23*

ties required to execute those commands, and the Object Manager provides the shared object storage.

The Session Manager component provides the mechanism by which an application starts a database session. It creates an instance of each of the Request Manager and the Data Manager for each database session.

Figure 5 also shows the main interfaces between the components of the system. The first is ESQL, which is used by an application to access the database system. Lera is an extended relational algebra that is used between the Request Manager and the Data Manager. Last is the Process Control Language interface provided by the kernel.

A set of ESQL commands forms the input to the Request Manager, which compiles these commands in five stages:

- *Syntax analysis.* This stage parses the input and converts it to an internal structure. It also performs type checks.
- *Logical optimization.* The logical optimizer reorganizes the query by applying transformation rules to the query. These transformations perform functions such as predicate migration to minimize the size of intermediate results, the elimination of common subexpressions to remove redundant work, operator transformation to combine operators to simplify the task of the physical optimizer, application of constraints, and optimization of recursive queries.
- *Physical optimization.* The physical optimizer determines the order of the basic operations to minimize intermediate results, selects the best access path, chooses the algorithms, and determines the optimal degree of parallelism in the query. The choice of these options is based on the minimization of a cost function.
- *Parallelization.* The parallelizer translates the intermediate form generated by the physical optimizer into the parallel program representing the query.
- *Code generation.* This stage performs the final generation of the object module containing machine code and calls to the runtime facilities of the Data Manager.

As Figure 5 shows, the Request Manager consists of four main components: the monitor, analyzer, compiler, and catalog manager. The monitor provides the operational interface between the application and its instance of the Request Manager. The analyzer performs the first stage of the compilation of a query, and the compiler performs stages 2 to 5.

To support the management of the relations in a database and the compilation and optimization of queries, the Request Manager maintains a catalog, sometimes called a metabase, of information about the relations and schema in the data-

base. The catalog manager provides the Request Manager with a simple interface for accessing this data.

The parallel programs generated by the Request Manager execute in the runtime environment provided by the Data Manager, which consists of four main components:

- *Relational Execution Model.* This runtime library includes relational operations; operations supporting the ADT, objects, and rules; and controls operators.
- *Relation Access Manager.* This manager provides a global abstraction of the relations in the database. That is, it hides the distributed nature of the relations from the operations in the Relational Execution Model. The manager also provides the mechanism for calling the appropriate access methods for the indexes associated with a relation.
- *A set of access methods.* These methods provide the mechanisms for accessing the tuples of a relation. One access method will implement each index associated with a relation. The indexes offer fast methods for accessing the tuples of a relation.
- *Basic Relational Execution Model.* This parallel program environment provides abstractions tailored for the efficient execution of Request Manager programs.

The Object Manager provides basic object storage and manipulation facilities required to support the database system. This manager stores persistent objects, controls concurrent use of shared objects, and provides logging and recovery facilities for transaction support. We based the Object Manager on the Arjuna system<sup>5</sup> with ideas incorporated from the CHOICES<sup>6</sup> and Camelot<sup>7</sup> projects.

**Parallel execution of queries.** The major influences on the Relational Execution Model design were the DDC project<sup>8</sup> and the Bubba project.<sup>9</sup> DDC was a project in the ESPRIT I program that had the objective of building a multiprocessor database machine. The Bubba project was a multiprocessor database system. We based the parallel execution of the queries on the following principles:

- 1) The relations are horizontally partitioned into fragments that are distributed across the set of available processing elements of the machine. One of the advances in the physical optimizer is the development of methods for determining the degree of parallelism in an operation. These methods allow the system to determine the optimal number of processing elements to be used during evaluation. For base relations the assignment of fragments to physical processing elements is relatively static. For the intermediate relations however, the assignment occurs at execution time. We refer to the set of processing elements across which a relation is partitioned as the *home* of the relation.

- 2) Where possible, processing takes place at the location of the data so the data is not moved. Naturally, this is not possible when an operation involves more than one relation. In this case the optimizer must choose the local operations in such a way as to minimize the movement of data.
- 3) The separation of data flow and control flow allows optimizations that significantly reduce the number of control messages.

A standard exemplar that is being used within the project forms the basis of the description of the parallel evaluation of queries. This exemplar is a share management system. The schema in Figure 6 defines two relations from the exemplar.

```
CREATE TABLE scost (
  share-id c4,
  cost integer2;
  currency c2);
CREATE TABLE exchange (
  currency c2,
  rate integer2);
```

and the query is:

```
SELECT share-id, cost, rate FROM scost, exchange
WHERE cost < 100 AND scost.currency = exchange.currency
```

Figure 6. Share management system exemplar.

We plan to extend the Create table command to allow users to specify the distribution algorithm, the attribute to be used, and the size of the home. In the absence of user-supplied information the physical optimizer chooses these parameters for the relations. The Data Manager determines the mapping of the relations based on the sizes of the relations' homes and the loading of the processing elements.

In this example we assume that the Data Manager chooses processing elements 1, 3, 8, and 9 for Scost and 4 and 5 for Exchange. We also assume that a hash function on Share-id distributes Scost, and a hash function on exchange distributes Currency.

When the Request Manager compiles the query, the physical optimizer decomposes the Join operation implied by the query into two suboperations Sel and Join. Sel prefilters the local fragment of Scost for those tuples in which cost is less than 100. Sel then distributes the tuples using the hash function for Exchange. The Join suboperation joins a tuple from Scost with the local fragment of Exchange. A trigger message sent to the Sel operations starts the processing of the query. Figure 7 illustrates the execution of this query.

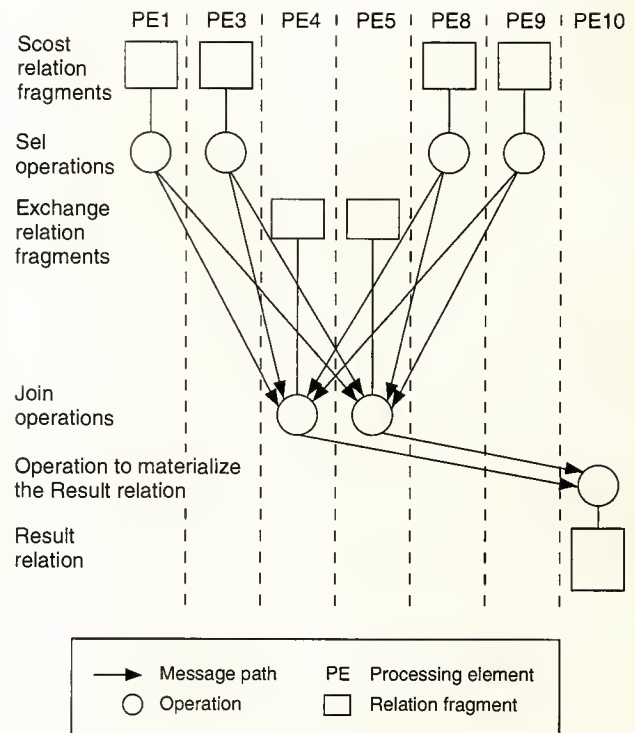


Figure 7. An example of a query execution.

This simple example illustrates the main principles of the computational model:

- relations are partitioned into fragments, which are distributed across their homes;
- relational operations decompose into operations that execute at the home of the relations on which they operate and so use purely local data; and
- their inputs are either a stream of messages or a fragment of a stored relation.

However, this very simplified account of the execution of the query does not discuss many important issues. One important benefit of processing the relations locally is that it allows the lock management to also be processed locally, thus providing an important performance improvement.

### The Elipsys language

Elipsys<sup>10</sup> is a parallel logic programming system for complex applications. The system integrates Or-parallelism, constraint satisfaction through finite domains, and an interface to the EDS database server.

The particular combination of Or-parallelism and constraint-satisfaction problem-solving techniques, which prune the



search space in an a-priori manner, provides an efficient platform for executing search-intensive programs. Elipsys solves a typical combinatorial search problem—for example, graph coloring, scheduling, and some other related operations research problems—in polynomial time.

The syntax of the Elipsys programming language is derived from DEC-10 Prolog. Elipsys makes available to the programmer the following features:

- *Data-driven computation.* This feature gives the programmer a flexible way of instructing the logic programming system in the way the paths of the search space can be computed.
- *Built-in constraints.* We build in simple equalities and inequalities, linear equations, and optimized branch-and-bound techniques, which range over the domain of finite discrete sets.
- *User-definable parallel constructs.* Predicates can be annotated as candidates for parallel evaluation and interface to the EDS database server through ESQ.

The Elipsys execution model in Figure 7 combines a message-passing architecture for the control and scheduling of parallel work and a distributed, shared virtual address space for the implementation of the binding environment. The above combination permits Elipsys to execute efficiently under the Emex kernel by taking advantage of the facilities provided for task and thread management. It also uses the Emex-provided, distributed, and shared virtual memory scheme, which is kept coherent by a "lazy, strong" method. This coherence scheme does not perform any coherence maintenance operations by default; explicit synchronization points in the application code trigger the operations.

The Elipsys binding environment is both read-only and shared. A control Or-tree and a shared environment represent the search space. A descendent Or-node inherits the binding environment of its ancestor Or-nodes. This inherited environment is read-only. Thus all the descendent Or-nodes hold the same view of the inherited environment. Modifications to the shared environment occur through auxiliary structures, which are local descendent Or-nodes. These structures in turn become shared whenever a control Or-node gives rise to any descendent Or-nodes.

The message-based Elipsys control and scheduling mechanisms make use of the control Or-tree data structure, which is distributed over a set of workers. Each worker is allocated to one EDS processing element; a worker consists of a distributed scheduler, performing scheduling and control functions, and a set of engines. An engine performs sequential resolution steps, extended linear resolution with a selection function applied to definite clauses over finite domains. It also manages the interface to the EDS database server. A scheduler-engine interface describes the schedul-

ing policy, pruning, and input/output interactions between the scheduler and the engine.

## Advanced applications using Elipsys

Elipsys is oriented toward complex applications. We are developing a suite of programs to demonstrate the practicality of Elipsys for a wide range of applications domains. These programs will highlight different design features of Elipsys:

- *Compatibility with existing applications.* A civil engineering program analyzes possible faults in concrete piles from acoustic data. The UK University of Bristol is parallelizing and porting this sequential Prolog program to the Elipsys subsystem. This activity will identify the potential problems that may be encountered when converting existing Prolog applications to Elipsys.
- *Capability for handling large data sets.* The University of Athens is developing a tourist advisory system for Greece. The system provides customized holiday packages for individual tourists as well as general information for potential visitors to Greece. This application makes extensive use of the Elipsys connection to the EDS database subsystem. The raw tourist data can be stored in the EDS database rather than within Elipsys itself.
- *Deductive capability.* System and Management S.p.A. in Italy currently implements a Treasury Management System, an expert system for banking. Its role is to suggest profitable investment plans to bankers. Such an application is well suited for Elipsys and fully uses Elipsys' inherent deductive power and capability for interworking with the EDS database server.
- *Capability for managing complex data structures.* ECRC in Munich is developing several applications in the domain of molecular biology. This domain requires the management of vast amounts of complex data, again using the Elipsys/database connection. Moreover, the system requires the complex symbolic processing, for example, in structures matching different DNA molecules, and the built-in constraint facility of Elipsys.

## The application language

Lisp is a powerful general-purpose programming language whose programs are written on a much higher level of abstraction than the Pascal, Ada, and C procedural languages. Being so powerful, Lisp has been widely used for complex applications such as artificial intelligence. To this expressive power, we added the power of parallel processing in EDS Lisp.<sup>11</sup>

EDS Lisp extends the Common Lisp language. Because Common Lisp constitutes a de facto standard, users can easily port most existing Lisp applications to the EDS machine.

We selected the extensions of EDS Lisp after an intensive

study of other parallel Lisp systems. The extensions allow access to the EDS database system, and they provide language constructs for explicit parallelism. Explicit parallelism enables the programmer to specify large-grain parallelism that fits well to distributed-memory machines like the EDS.

An EDS Lisp program can have an indefinite number of parallel processes. The programmer creates processes to perform some action in parallel and to return a value. The EDS system schedules these processes. EDS Lisp contains a single construct to spawn parallel processes, the Future construct known from other parallel Lisp dialects.<sup>12</sup> Future constructs support transparent use of results of parallel processes. The main idea is that a Future immediately returns an (initially empty) placeholder for the result of the spawned process. The spawning process can then continue operation. When some process accesses this result, it waits until the result is available and then continues operation. Both the placeholder mechanism and the implicit waiting are invisible to the programmer. Consider, for example, the following piece of EDS Lisp code:

```
(setq x (future f p1 ... pn))
```

which corresponds to

```
x := future (f (p1, ..., pn));
```

in a procedural programming style. A parallel process is spawned using the Future construct to compute the function  $f$  with parameters  $p1, \dots, pn$ . The Future call immediately returns a placeholder for the result of  $f$  and assigns it to the variable  $x$ . The spawning process then continues in parallel to the process computing  $f$ . If a process reads the variable  $x$ , it tests implicitly whether the result is available and waits if necessary.

EDS Lisp also provides a Mailbox concept for communication between processes and a Critical Section mechanism, among other things, for synchronized access to shared variables.

## Metal

The Metal machine translation system translates natural-language documents<sup>13</sup> and currently requires a special-purpose Lisp machine for production use. The EDS Lisp application is complex enough to conquer both the CPU-power and storage limitations of today's workstations.

We expect a speedup of more than a factor of 300 for running Metal on the EDS machine; the translation of 250 pages that needs 10 hours today should be accomplished in two minutes on the EDS machine, as shown in Figure 8. This performance increase is highly relevant for the application, because the translation volume for technical documentation

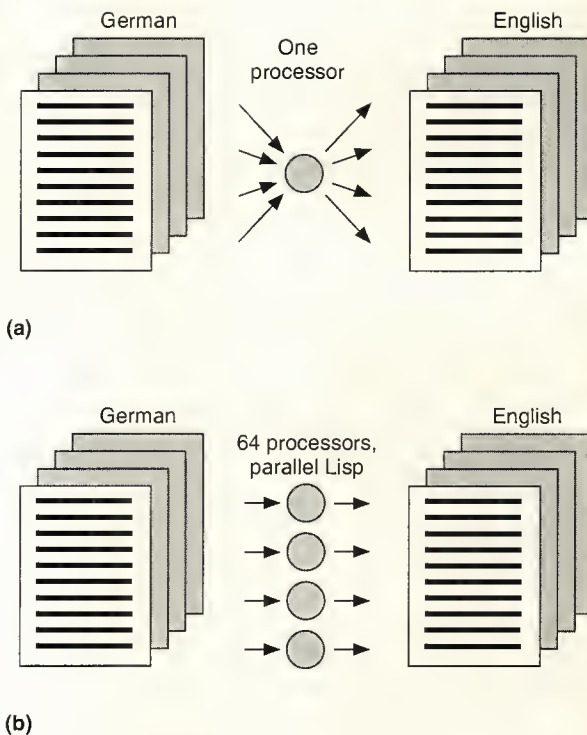


Figure 8. Sequential (a) and parallel (b) translation on Lisp systems.


is immense. The documentation for a complete technical product line often amounts to several hundred thousand pages that have to be made available in a multitude of languages.

Users can access the parallel processing power of EDS Lisp not only to translate such a huge mass of text but also to improve the quality of the translation by using more advanced and therefore more resource-intensive algorithms.

THE EDS PROJECT IS A MAJOR, PROMISING Commission of the European Community-sponsored ESPRIT II collaboration between Bull, ICL, Siemens, and their jointly owned ECRC research center. The EDS system primarily focuses on the large-scale information server, which must manage information efficiently and effectively across the spectrum from data to knowledge.

The EDS system enables programs calling the SQL, Lisp, and Elipsys interfaces to exploit large-scale parallelism essentially transparently. We've described the database and language aspects of the EDS system, looking at the SQL, Lisp, and Elipsys subsystems.



The EDS project combines the complementary skills of its partners and associate partners to achieve a clear and common goal. At the end of the second of four years, the project is on schedule to switch on an EDS machine in 1991 and demonstrate applications in 1992. 

## Acknowledgments

We thank both the European Commission, DG XIII, and the senior management of the EDS partners and associate partners for their continuing support for this strategically important European initiative. We also thank all the members of the EDS team for their dedicated work on the project.

## References

1. G. Haworth, "Information Servers, Present and Future," *Proc. Unicom Commercial Parallel Processing Seminar*, Unicom, Uxbridge, Middlesex, UK, to be published in 1991.
2. ISO 9075, *International Standard, Information Processing Systems, Database Language SQL*, 1987-06-15, International Standards Organization.
3. G. Gardarin and P. Valduriez, "ESQL: An Extended SQL with Object and Deductive Capabilities," *Proc. Int'l. Conf. Database and Expert System Applications*, A.M. Tjoa and R. Wagner, eds., Springer-Verlag, Berlin, Aug. 1990, pp. 299-307.
4. M. Ward, P. Townsend, and G. Watzlawik, "EDS Hardware Architecture," *Proc. Parle 90 Conf., Lecture Notes in Computer Science*, H. Burkhard, ed., Vol. 457, Springer-Verlag, pp. 816-827.
5. G.N. Dixon and S.K. Shrivastava, "Exploiting Type-Inheritance Facilities to Implement Recoverability in Object-Based Systems," *Proc. Sixth Symp. Reliability in Distributed Software and Database Systems*, Mar. 1987, pp. 107-114.
6. R. Campbell, G. Johnston, and V. Russo, "Choices(Class Hierarchic Open Interface for Custom Embedded Systems," *ACM Operating Systems Review*, Vol. 21, No. 3, July 1987, pp. 9-17.
7. A.Z. Spector et al., "The Camelot Project," *Database Engineering*, Vol. 9, No. 4, Dec. 1986.
8. B. Bergsten and R. Gonzalez-Rubio, "A Database Accelerator and Its Languages," *Proc. Conpar 88, 8CS Workshop Series*, C.R. Jesshope and K.D. Reinartz, eds., Cambridge University Press, 1989, pp. 63-71.
9. H. Boral et al., "Prototyping Bubba, a Highly Parallel Database System," *IEEE Trans. Knowledge and Data Engineering*, Vol. 2, No. 1, Mar. 1990, pp. 4-24.
10. S.A. Delgado-Rannauro et al., "A Shared Environment Parallel Logic Programming System on Distributed Memory Architectures," ECRC document submitted to 2nd European Distributed Memory Computing Conference, (available from author), 1991.
11. C. Hammer and T. Henties, "Parallel Lisp for a Distributed Memory Machine," *Proc. Lisp Europal Workshop on High Performance and Parallel Computing in Lisp*, Ashok Gupta (ed.), Europal, Surrey, UK, Nov. 1990.
12. R. Halstead, "Multilisp: A Language for Concurrent Symbolic Computation," *ACM Trans. Programming Languages and Systems*, New York, Vol. 7, No. 4, Oct. 1985, pp. 501-538.
13. T. Schneider, "The METAL System, Status 1987," *Proc. Machine Translation Summit II*, C. Rohrer and T. Schneider (ed.), Deutsche Gesellschaft fur Dokumentation, Frankfurt, 1989, pp. 128-136.



**Guy Haworth** is the marketing manager for Information Servers within ICL's Computer Products Division. He has worked with ICL's database and CASE products, particularly the CAFS (Content Addressable File Store) search engine and is now responsible for the market focus and exploitation of the EDS project. He is interested in the performance and reliability of large-scale database systems.

Haworth holds an MA degree in mathematics from Oxford University and the Diploma in computer science from Cambridge University.



**Steve Leunig**, ICL's lead designer on the EDS database system, has worked as a programmer and lead designer on its VME operating system. Most recently, he participated in the Alvey Programme's Flagship Project, which developed a multiprocessor graph-reduction machine.

Leunig holds a BSc degree in mathematics from the University of Western Australia and an MSc degree in computing science from the University of London.



**Carsten Hammer** works in Siemens Corporate Research and Development in Munich. He leads the Languages for Parallel Computers Group and holds project responsibilities for the parallel EDS Lisp system. Previously, he led the activities for a vectorizing Pascal compiler.

Hammer received the Diploma in computer science and his doctorate from the Technical University of Braunschweig in Germany.



**Mike Reeve** currently leads the Computer Architecture Group at the European Computer Research Centre GmbH (ECRC). The group researches parallel declarative programming, distributed operating systems, and sequential and parallel machine architectures. In 1989 he accepted the Sony

sabbatical chair, working at Sony's Computer Science Laboratory in Tokyo. For the five preceding years he held a SERC Advanced Research Fellowship at Imperial College, University of London.

Reeve obtained his BSc and PhD degrees in computer science from Imperial College.

Direct questions regarding this article to Guy Haworth, Marketing Manager, Information Servers, ICL, Kings House, 33 Kings Road, Reading, Berkshire, United Kingdom RG1 3PX.

### Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159

Medium 160

High 161

### SPAN

*continued from p. 27*

in which the `||` operator in example i) specifies that statements  $s_1$  to  $s_n$  are to be performed in parallel. On one processor, this operation may be mapped into a series of processes activated in some undefined order. In ii), the `;` operator specifies sequential execution of the statements. Replicators exist for simplifying repetitive parallel or sequential statements. Examples iii) and iv) are alternatives to i) and ii). Parle only provides for synchronous execution of processes initiated by the `||` operator, in that these processes must all terminate before the initiating program can continue to the next statement.

The simplest conditional statement contains a single statement, which is executed when a guard condition is true. The most complex conditional statement contains multiple, guarded statements; the guards execute in parallel. When the statement associated with the first true guard executes, partially executed guards are discarded.

Although intended as a compiler target language (CTL), Parle is probably a better parallel programming language. When it is used as a CTL, Parle's weak typing places a large runtime checking burden on architectures not specifically designed to support the architectural model. The process model also has limitations, especially the limited control over processes and their synchronous execution. Process creation is statically defined at compilation time, and processes once initiated run to completion. The model cannot suspend or delete a process once it is started because there is no process by which an executing process can be identified. Neither does the model support dynamic process creation and process migration.

The process model is fine for a programming language, but it limits a CTL. The applications projects produced substantial amounts of code in Parle, which was easy to use and effective.

### Virtual Machine Code

The VMC fully realizes the Kernel System model at the level of machine code or assembler. It provides a model of the Kernel System that can be ported onto a variety of parallel architectures. Consequently, the VMC is a low-level language that reduces the work of performing the port. The philosophy behind the design of the VMC was again to provide the necessary components to support the Kernel System while limiting complexity. Thus, the VMC has

- a reduced instruction-set style computational model.
- a small, simple instruction set with which more complex operations can be implemented. (The exception is that list operations are members of the instruction set, as lists are primitive elements.)



- only one addressing mode.
- a load/store philosophy extended by message-passing operators get and put.

We assumed that all instructions would take equal time to execute (of course, this assumption cannot be maintained for list operators and nonlocal accesses at the hardware level).

The VMC has a flat, unbounded address space of processors identified by number: 1 to  $n$ . Each processor has a local list-structured memory and a root pointer (called gp) to the top-level list. (See Figure 2 again.) A processor can randomly access any location within its local memory by address. An address is a list containing a starting point, or context, in the memory. It is followed by a sequence of selectors that specify how to traverse the memory from the context to the addressed memory element (for example, the five-element address list [gp 4 4 3 2] in Figure 2). Using gp as the context for an address provides an absolute addressing scheme. Other contexts can provide for relative addressing.

A processor can access the memory local to any other processor through an extension of the addressing scheme. Thus, [gp 0 45 3 2 4] is the address of a memory element in processor 45. The 0 selector applied to gp is an escape mechanism to proceed from local memory into the address space of processors in which the following selector is used to identify the processor. Nonlocal memory locations can only be absolutely addressed. Note that the first element in all lists has the selector 1.

The VMC provides for a fully list-structured, von Neumann memory with lists, integers, and the empty state as primitive data types. Shared memory and message-passing occur through the four memory operators (load, store, get, and put) in both local and nonlocal memory. The VMC maps both code and data into lists and manipulates them accordingly, as it also does for addresses.

The VMC provides the standard arithmetic and logical operators for integers. It also provides a basic set of list operators similar to those in Parle. These operators

- find the length of a list,
- concatenate two lists,
- form a sublist (either starting from the head of the list or ending at the tail of a list), or
- create a new list.

The list operators provide symbolic operations, while the addressing mechanism and memory operators provide the random-access capability required for numeric operations.

The VMC has a control-flow model of execution. Its program counter identifies both the memory list and the element within this list from which the next instruction is to be fetched. Unconditional and conditional branch instructions allow for branching within the current code list or into another code

list within the processor's local memory. Each processor has an evaluation stack (similar to that of the transputer). The VMC places data values on this stack when they are fetched from memory. Operators find their operands on the stack and return their results to that location. The VMC does not have the usual, numeric stack with unlimited depth. The return information from subroutine calls has to be explicitly moved from the evaluation stack into memory elements at the start of a subroutine.

We wanted to again limit the complexity and give flexibility to system implementers in the handling of subroutine calls. However, the system needed a local data space for subroutine parameters, local subroutine variables, and return addresses—and a relative addressing scheme to access the space. In particular, the system had to support recursive subroutines. The VMC provides for these needs by adding extra contexts that can identify sublists in the local memory. The sublist selection can be changed as required, say, on subroutine calls, to provide a new list for local data.

The VMC implements copy semantics as described earlier. When a load is executed, the system copies the contents of the addressed element. It then places the copy (whether list, integer, or empty) on the execution stack. When a store occurs, the item (list, integer, or empty) that is overwritten in the addressed memory element is destroyed. Thus, no sharing of lists occurs in the VMC. The VMC does not require garbage collection for lists because they are immediately destroyed when overwritten or otherwise consumed. Although this mechanism simplifies the VMC, it has major implications for hardware supporting the VMC. During a write, for example, the system reads the value to be overwritten and checks it to see whether it is a list. If it is, the system deletes the list and frees the memory it occupies.

The VMC has a simple multiprocessing model for process selection and execution. Only one process list exists. When a process is suspended—either through the suspend operator or a blocked message-passing operation—the executing process takes its place at the end of the process list. The first process on the list is removed and executed. When a process spawns a new process, the current process is placed at the start of the process list and the new process begins execution. No access is provided to the process list, and one cannot change the sequence of execution.

The parallel processing model of the VMC allows processors to operate independently and intercommunicate via operations on nonlocal memory. The VMC specifies nothing else about the communications (no specified communications hardware, protocol, or message structure exist). The VMC model of interprocessor communications is strictly one of accessing a memory element. The VMC passes the content of a memory element (list, integer, or empty). No restriction exists as to the complexity of a list that is being passed. (The model can copy all of a computing node's local memory by

performing a load operation at a processor's address.)

An example of VMC code that concatenates two lists and stores the result follows:

[gp 1 5 6 88]	Load address onto evaluation stack
load	Perform load on element, copying contents to evaluation stack; Address is consumed and removed from stack
[gp 63 7 12]	Load address onto evaluation stack
load	Perform load on element, copying contents to evaluation stack; Address is consumed and removed from stack
cat	Concatenate two items from top of stack, return result list to stack; Consume operands—error if not both lists
[gp 9 12 3 6 5]	Load address onto evaluation stack
store	Store list into memory, address and list are removed from stack

The major problem with the VMC is that its low-level nature makes it difficult to map onto most existing architectures. This characteristic also defeats its primary objective: easy porting of the Kernel System to various machines. However, the VMC does provide a very powerful and consistent architectural model that fully supports all the features of the Kernel System. In addition, its low-level nature (conversely) makes it an excellent architectural model for the development of a computer system especially targeted to support the Kernel System. Thus, the VMC became the model for the VLSI work package of SPAN that developed the Sprint processor described in the following section.

### Sprint architecture

The aim of the VLSI work package was to design a parallel computer system that efficiently supported the Kernel System. We further constrained the work package to support the VMC, for a number of reasons. The VMC provided an excellent model to work with, and its low-level nature restricted the search space of the designers. The VMC's reduced instruction set also fit well with the design and development approaches feasible for a team of three.<sup>6</sup>

We targeted the design not at a machine that directly modeled the VMC, but to a functionally equivalent machine to which VMC programs could be easily mapped. Also, we did not expect that all the VMC features would be directly supported by hardware. (In fact, the hardware does not support the list creation, extension, deletion, and comparison operations.) Our primary design objectives were to support most of the features of the VMC model as efficiently as possible. These features included its list-structured, von Neumann memory with shared memory and message-passing

operators, and an operand stack.

We hoped to obtain as much hardware support as possible and reasonable with a team of three and a limited budget. The design allowed those features not directly mapped into hardware to be easily and efficiently mapped into software.

Our secondary objectives were to expand the design (without restricting any of the primary objectives) to make it as general purpose as possible. We wanted to

- support different programming styles,
- provide an architectural environment suitable for a complex operating system (such as protection, flexible task scheduling, interrupt support, etc.), and
- provide a robust environment to limit the effects of incorrect programs (such as hardware-type checking of instruction operands and hardware-bounds checking on a list access that ensures the access is within the length of the list.)

An important overall design objective (especially to the designer) was efficient, fast operation. To this end, we concentrated on minimizing the following:

- the processor clock period,
- the number of clock cycles per instruction,
- the memory-access time, and
- the number of these accesses.

These elements are particularly important with respect to the implementation of list-structured memory.

We initially divided the design of the computing node into processor, list-structured memory, and communications. We developed a novel algorithm to support the list-structured memory so that it could be implemented within the processor. Thus, we only needed to develop two components for the computing node: a communications chip and a reduced instruction-set computer chip.<sup>6,7</sup>

**List-structured memory.** The implementation of the list-structured memory of the VMC was a major issue. It affected the development of many other parts of the project. This implementation is central to the provision of a computer that efficiently executes VMC programs and serves as a fast engine for both symbolic and numeric processing. In particular, it was necessary to furnish a true random-access capability for such numeric processing as array and structure accessing, and instruction branching. We looked for a list-memory scheme that could be implemented on a standard, linearly addressed, physical memory (readily available at low cost and high densities). By introducing a level of indirection, the level of a memory element is no longer required to hold a list. Instead, a memory element only needs the capability of holding a pointer to a list. The major memory problem remains the structure to which the pointer refers.



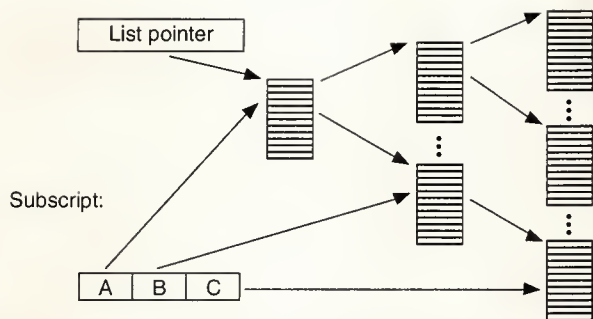


Figure 3. Mapping a list into physical memory pages.

We developed a novel page-based scheme from knowledge of page-based memory management.<sup>8</sup> We chose a paged scheme for the same reason it is used in memory management: the ease of free space management in a system with dynamic memory allocation. The tree-structured scheme supports random access to all list entries. It also supports list extension at the head and tail of the structure without having to copy the list. Each list requires its own instance of a tree structure. Its memory elements consist of the leaves of the tree, grouped into pages. The branch nodes in the tree are page pointers down the tree structure, also grouped into pages (see Figure 3). At the top of the tree is a single page with pointers to lower level pages. The number of levels between the top of the tree and the leaves depends on the list size and the page size. The tree structure is referenced by a list pointer, which contains the address of the top-level page, the length of the list, and the depth of the tree structure. The architecture provides for the existence of only one list pointer to a tree structure because:

- the list length is stored in the pointer,
- the system uses on-the-fly garbage collection, and
- the list-structured memory model does not support the merging of the memory structure as would occur with storage of multiple copies of a list pointer.

One such tree structure represents each list, although the number of levels in the structure varies with the list length. Consideration of the memory element size led to restrictions on the maximum list structure and the maximum list length. We selected a page size of 32 elements. A larger size would have led to fragmentation, since many lists will be short. A

smaller size would have led to deeper structures and more memory accesses. The page size, however, did lead to a restriction of a maximum of three levels in the list structure and a maximum list length of 32,768 elements. This arrangement limits to 15 bits the subscript needed to locate an entry in a list. An entry is located as follows:

- divide the subscript into three 5-bit fields (A, B, and C in Figure 3),
- use field A to select an entry in the first-level page,
- use the address produced with field B to select an entry in a second-level page, and
- use the address produced with field C to select an entry in a third-level page.

**Tagged architecture.** We created different data types by placing a tag on each and every memory location. This enabled the three VMC data types—integer, list, and empty—to be correctly identified. The decision to have a tagged architecture greatly influenced the further development of the design. It allowed the separation of list pointers from integers and runtime hardware type-checking of instruction operands with no overhead. This method also enabled a number of other mechanisms to be implemented in hardware to provide efficient coding and reduced runtime overhead.

Tagging required a larger memory element size than 32 bits, since a 32-bit integer capability was deemed essential. The components included in the list pointer—page address, length, and level structure—further increased the size of the memory element to 40 bits. The level structure, combined with the tag, produced a 3-bit tag field and eight data types. This provided list data types for one, two, and three-level list structures, in addition to integer, empty, and three extra data types. One of the extra types marks the first element in an address list—the reference point—so that address lists can be identified from data lists to provide runtime checking of addresses. The other two extra types are reserved for users. Hardware performs conditional checks on these types, among others.

Figure 4 shows the structure of the list pointer. The provision of a tag field and a number of data types led to a more robust environment through runtime checks on data objects, addresses, and instruction operands. This approach reduced the complexity of both the processor and data management by making the data types distinct. No operations to the data field of any object could change it into a different data type. The tag field also helped simplify and shorten program code.

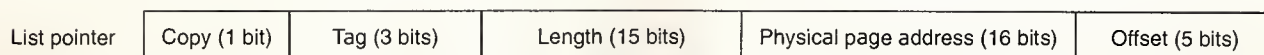


Figure 4. Structure of the list pointer.

Using only a 3-bit tag limits the hardware expense of providing the runtime checking facilities.

**Modes of operation.** Early in the design, we saw that many VMC operations could not be easily implemented in hardware and should be supported in software. These tasks included complex list operations (except obtaining the list length), multiply and divide (because of the silicon area required), task scheduling, and register save/restore operations on task-switching. The system also had to support exception handling for such functions as blocking put and get operations and process-switching, list-bound checking exceptions, instruction tracing, and interrupt handling. These requirements led to the provision of three modes of operation: executive, supervisor, and user.

In the executive mode, which accesses the whole system, the system inhibits all exceptions and interrupts. The major roles of the executive mode are to handle task-switching—whether generated by hardware or software exception, or by interrupt—and to organize the task scheduling and initiation. Software must perform the latter to give greater flexibility to the scheduling algorithm.

The processor contains a timer register with an interrupt signal to allow for time-sliced scheduling. The executive mode has its own set of registers, including code pointer registers. Consequently, no state must be saved by either software or hardware on entry to executive mode. This arrangement simplifies the hardware design and provides a fast switch to and from the mode, which can execute very short sequences of code without interruption or large overhead.

The fact that no interruption of executive mode can occur limits the code that can execute in it and requires that code to be fully tested and reliable. Thus, the executive mode serves key functions only.

To provide for the rest of the operating system, the nonexecutive mode subdivides into supervisor and user modes. These modes are identical in terms of the accessible instructions and registers, differing in just two ways. First, interrupts can be masked in supervisor mode only. Second, the software-exception vectors generated by the two modes subdivide into two disjoint sets—even vectors to user mode and odd vectors to supervisor mode. Because one mode cannot generate the software-exception vectors of the other, the modes are completely separate. Each mode enters the executive mode at a different point, which obviates privilege checks before code execution to simplify and speed operation.

We used the odd/even divide operation to locate all the entry points close together at the start of a list in the event of a small numbers of vectors. We envisaged an operating system that would mainly execute as tasks in supervisor mode. These tasks would be scheduled in the same way as user tasks. The system would enter executive mode to execute short sections of essential code, select a new task to execute, and switch back to nonexecutive mode.

The hardware allows for up to 16K software vectors for each mode, although the executive mode sets the maximum that can be generated upon entering the nonexecutive mode. This large number of software vectors provided great flexibility and was easy to implement with hardware already present for other purposes.

**Instructions and registers.** Access to elements in the memory requires the traversal of the list memory structure through translation of logical list addresses such as [gp 3 5 2] to physical memory addresses. We designed the processor to perform this translation process as efficiently as possible. The processor has 32 instructions, three of which are multicycle instructions that perform address translation. The most complex of these instructions translates a full logical address in an indivisible sequence to locate and read the addressed memory location. We see these multicycle instructions as essential to keeping the memory access times as short as possible and to maintaining memory consistency throughout the address translation process. They are developed on top of the single-cycle instructions, which act as a form of microcode. Consequently the multicycle instructions do not compromise the processor speed and only add a small amount to chip size and complexity.

We used a small register set to restrict the process state and the physical layout. The size of the physical layout prevented the use of register windows. (The architectural model did not suit their use, anyway). We employed 28 addressable registers and 32 register references, since the latter do not all refer to different or distinct registers. A subset of 16 registers is exclusively available for the nonexecutive mode. Four of the general registers can function as either ordinary registers or a four-element stack used by the VMC. Two register references allow an item to be pushed or popped from the stack. The flags register holds information about the current top of stack and the stack depth. Four other register references allow any of the four stack elements to be accessed if it exists. All stack violations cause a hardware exception. If the stack size is set to a maximum and the push and pop references are not used, the registers can function normally.

We adopted a 16-bit instruction size, which led to the choice of 32 instructions and 32 register names. This approach enabled two instructions, called an instruction pair, to reside in one memory element. A 4-bit condition code that further controlled the execution and interpretation of the instruction pair accompanied it. One can use 5- and 16-bit literals (constants) with any instruction. In the latter case, the literal replaces the second instruction of a pair. Because the instruction set and register set are orthogonal, no hardware restrictions exist for their use—although some combinations are not sensible programming.

Placing two instructions into a memory element allowed us to improve processor operation by making the execution of the pair indivisible by interrupts, bus arbitration re-



quests, and shared bus accesses. Both hardware and software exceptions are taken between the instructions of a pair. This mechanism treats the instruction pair almost like the single instruction in more traditional architectures, giving a somewhat greater complexity to the instruction set. The initial and major reason for doing this was to provide indivisible memory operations. These operations allow a list address to be evaluated and a memory element to be read and modified (necessary for get and put) without any possibility of invalidation by some other memory operation. (Operations via direct memory access or task-switching could otherwise invalidate the address translation or modify the addressed memory element under some circumstances.)

The instruction-pair mechanism allows memory operations to be handled by an address-translation instruction, followed by a second data-movement instruction. The address-translation instruction locates and reads the addressed memory. The second instruction moves the read data to a register or writes a new value to memory. All memory operations include an initial read of the addressed memory, followed by a second instruction to move the read value to a register or to perform a write to memory.

Our approach simplified and reduced the necessary instructions and their complexity, while still allowing a more powerful set of operations to be performed via instruction-pair combinations.

The instruction pair also allows for an alternative form of conditional branching that doesn't break the instruction-fetch pipeline. This test-and-skip instruction either conditionally executes the second instruction or skips over it. The test-and-skip instruction and the more usual conditional branch instruction have a combined repertoire of 32 condition tests. Consequently, we kept these instructions even when a more powerful technique of conditional instruction execution was developed.

This technique uses the 4 spare bits in the integer element to provide 16 tests on whether to execute all or part of the associated instruction pair. The tests provide for executing both instructions of the pair or only the first one (for instances in which a no-operation instruction and a wasted cycle are required). More important, this technique can treat the instruction as a 32-bit literal and load it onto the evaluation stack without execution, which is useful for bringing in 32-bit masking patterns. The remaining tests are all conditionals on various flags or tag fields. This capability has proved useful in creating nonblocking put and get instructions. It enables the blocking action to be performed in conditionally executed software (a single instruction pair) with minimal overhead.

The tagged architecture also allows noninteger data objects to be placed in the instruction stream. The system identifies these objects by their tags and moves them to the top of the evaluation stack. Instructions are tagged as inte-

gers. This process requires trivial amounts of hardware and speeds the loading of these objects. It is particularly useful for fetching list addresses for memory operations. One can just insert the list pointers into the code.

**Addressing.** We intend user mode programs (and the bulk of any operating system software) to use the list addressing scheme exclusively. This approach allows physical addresses to be formed and consumed within indivisible operations so that memory consistency is rigorously maintained. Because of the dynamic memory structure, if a list is copied or even extended, the mapping of all list locations—and possibly of sublists—to physical memory can change. Thus no physical addresses, except within a list pointer, should be a part of the program state. Even on a subroutine call, the saved value for return should be a list address. (The actual value is programmable, since no subroutine call exists and the system has to build it on top of branch instructions.) The executive software makes use of both address forms.

The use of list addresses even extends to the code pointer registers. A list pointer register and a selector register identify the current code list and the next instruction within this list. However, to speed up sequential program fetching, a register that holds the physical address of the next instruction is kept consistent with the code pointer registers. In many instances, the system can use a physical address from this register, which removes the need for a list translation and reduces overhead.

Sprint has an expanded number of contexts. The architecture allows any register that can hold a list pointer to act as a context in a list address. This function provides greater flexibility and a choice of absolute or relocatable addressing for different compilation models. It also allows the movement of tasks within or between processors. Sprint provides absolute addressing by referencing all addresses from gp. Relative addressing requires the assignment of some fixed point or points in a program.

Figure 5 on the next page shows how a program in the C programming language might be mapped into a list structure as a single list with sublists. All global objects, data, and functions reside in the main program list (called program list in Figure 5.) This program could be stored in any location in the list structure memory of a processor. A relative addressing scheme would place the list pointer to the top-level list in a register. Sprint provides a register sd for this purpose (not shown in the figure). All locations within the program can be addressed relative to this. For instance, [sd 88] can reference some global data object, while [sd 44 15] can locate an entry point in a subroutine. Provided that physical addresses are not saved in memory, the program can move to any part of memory—even to another processor—when the program itself is not executing.

**Copy semantics and list deletion.** We designed Sprint

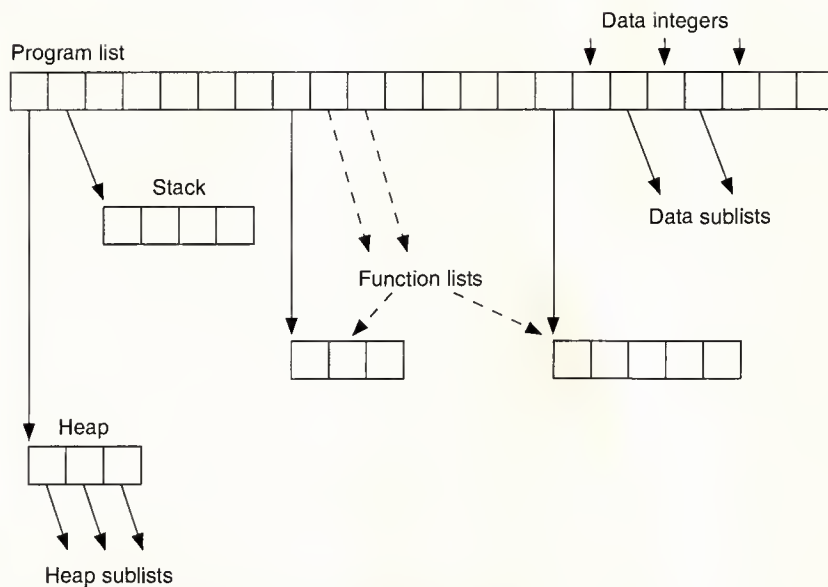


Figure 5. C-program structure mapped into lists.

to support copy semantics with two forms. The copy-on-load form of the VMC copies a list as soon as it is brought from memory by a load operation. A copy-on-store form delays the copying of a list until it is absolutely necessary, which reduces the amount of copying. For example, the latter form would not copy a list that is read into the processor just to find its length. Hardware exceptions support both of these forms. For copy-on-store, the system also provides a copy bit in the list pointer to record that a list should be copied when it is stored to memory.

If a write to memory overwrites a list pointer, the system generates an exception so that the underlying list can be discarded to free its pages. No other form of garbage collection occurs.

**Production.** The processor was manufactured on a 2-micrometer silicon chip with a total area of 70 sq mm. It subdivides into a data path with all the registers and functional units, and a control part made up of finite-state machines and instruction decoders. We designed the processor to operate from a 10-MHz clock with a maximum instruction execution rate of 5 MIPS (million instructions per second). The processor pipelines instruction prefetch cycles with instruction execution. Memory accesses can execute in one instruction cycle. We haven't completed testing the device. Further details on the hardware implementation appear elsewhere.<sup>6,9</sup>

**Communications.** The bulk of project activity concerned the processor, while the remaining effort went into developing a communications system. The aim here was to develop a simple, regular, extensible system around a switch

component—all on one chip. We looked for a parallel transmission system rather than a serial one. We rejected bus-based systems on the basis that a multiple-bus system was too complex and a single-bus system would limit the number of processors.

We developed a message-passing network similar to many packet-switching networks—except that the packet length is variable and unlimited. We designed a linear network around a single, three-interface switch element. The processor connects to one interface, while the other two interfaces connect to similar switches. Each interface channel contains two unidirectional, 11-bit channels: one for transmission and one for reception. Each channel operates independently, and the system is free of deadlocks. Single-item buffering occurs on each channel, and the system spreads a transmitting message over

the network. The interface between the switch and the local processor is memory mapped into the processor's physical address space. The interface automatically performs translations between 40-bit processor words and the 11-bit format used on the network.

Because the linear network is not an optimal choice for a large system, a two-dimensional grid system was our first choice. We chose the linear network, however, because it was easy to develop and because of pinout limitations on the final design.

We achieved all the goals of the VLSI work package. We implemented a design around the VMC that provides a consistent list-structured memory for both code and data with random access to any element in the memory. The design is fast, although not yet at the level of recent 20-plus-MIPS machines. Memory access speeds, although not always as fast as for a traditional memory, are still very good. The architecture will fully support an operating system and is broad enough to allow a variety of programming models to be mapped onto it. We found the use of a tagged architecture to be an especially decisive factor in producing a system with low overhead and improved reliability and versatility. In particular, tagging—along with the complex exception system—provides reduced processing overhead on function-operand checking and special processing for particular data types.

### The DICE architecture

The motivations for the development of the DICE architecture are somewhat unusual. Most parallel architectures



are geared for the fastest execution of a single program (with several processes). They offer all the available processors on an equal basis for concurrent execution. Instead, DICE is mainly oriented to the user, dedicating several processors to each user and giving absolute priority to that user's processors. We refer to these philosophies as program parallelism and user parallelism.

Many parallel architectures are designed for batch processing with the execution of one program. This program usually consists of several modules (each with several processes). Although the modules may be compiled separately, they are bound and allocated to processors in a combined way. This is program parallelism.

Many problems require this type of architecture. However, demand is increasing for highly interactive, general-purpose, multiuser computers that people can use daily for all types of tasks. A growing trend is the use of workstations, interconnected in a local area network, whose main advantage is to provide each user with one dedicated processor. This is user parallelism.

Making both types of parallelism available without having to simulate user parallelism is a fundamental feature in any successful, parallel, multiuser computer. However, parallel architectures with unshared memory have not reached this level of sophistication.

The main goal of the DICE distributed-memory architecture is to uniformly concentrate the functions of several networked workstations and/or program-oriented parallel computers in one parallel machine. In this approach, each user can have several dedicated processors rather than just one. The fact that interprocessor communication is much faster than in a local area network efficiently exploits program parallelism as well.

The computing environment must be as dynamic as that of current shared-memory computers, eliminating the traditional view of a host controlling an array of load-and-go processors. Each processor must have a resident kernel that provides the traditional services found in a uniprocessor. The kernel must also provide, among other things,

- interprocessor communication,
- dynamic load balancing involving automatic data and code migration, and
- a distributed virtual memory system with support for multiple copies (in different processors) of a given memory area.

**Incremental extensibility.** Parallel architectures cannot have a fixed configuration. The number of resources in the system and their configuration must be set according to the user's needs, which naturally vary with time. On the other hand, the increment (minimum change) should be very small to avoid discontinuities in the investment. Machines based

on hypercube configurations that contain power-of-2 numbers of processors are not incrementally extensible. The number of processors should increase according to the growth of the number of users or of their needs, not in powers of 2.

**Software/hardware independence.** Extensibility must not affect existing software. A machine in which the addition of a processor implies recompilation becomes awkward in a multiuser environment. The software (machine code) must run unchanged in every configuration, from the smallest to the largest, using whatever resources happen to be available. The software should also discover these resources automatically and dynamically. This independence, coupled with the extensibility, means that all configurations—from the PC to the mainframe—must run exactly the same code.

**System organization.** Interconnecting networks such as the hypercube have very limited extensibility. Solutions such as a tree or a mesh are preferable from this point of view, because the fixed number of connections per processor does not depend on the number of processors in the architecture.

Given the preference for user parallelism and extensibility, the DICE architecture adopted the tree as the interconnection topology. If a system has a small number of processors (generally up to 16), a bus serves as an extensible network par excellence. All processors connect from a minimum distance and can be added or removed without affecting the others. When local caches cannot avoid bus saturation, a higher number of processors can be organized into groups. They can connect to a communication element (CE) that then connects to an upper bus, forming an N-ary tree. The tree does not have to be balanced. Figure 6 on the next page depicts a typical configuration.

Besides permitting extensibility, the tree provides:

- favored local communication within each group,
- easy broadcasting of messages,
- simple message-routing performed in a decentralized manner locally to each CE, and
- a natural hierarchy. For instance, all processors in a subtree can share a disk located at the apex of that subtree.

Of course, traffic congestion occurs near the apex of the tree. However—due to the preference for user parallelism—this should constitute no real problem. Most of the traffic is local to each subtree. A natural organization is to assign a group of processors (headed by a CE) to each user, allowing the user to resort to a neighbor's processors whenever they are unused. Resources (which appear in the tree as disk 1 and disk 2 processors with special capabilities) that are permanently shared by several users are located in upper levels in the tree. Disks and printers fall into this category. The greater the number of users who share a given resource, the higher its position in the tree and the smaller

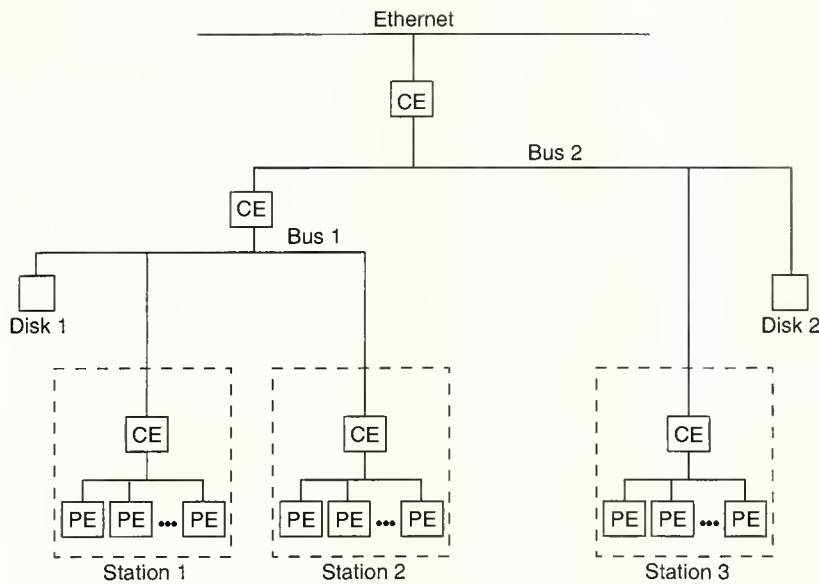


Figure 6. Typical configuration of the DICE architecture.

its traffic with each user. This way, the traffic near the apex of the tree is not substantially higher than that on the lower levels. This condition differs from program parallelism, in which all processors and resources communicate with all others in the tree in a uniform manner.

Each dashed square, called a station, includes the hardware dedicated to each user. This hardware includes one or more processing elements (PEs) and a CE that connects to the rest of the system. A station (in the context of the DICE architecture) is a board or a group of boards sharing the same rack with other stations (not one system connected to others by a local area network). A PE includes a processor and its private memory, but one could build a station with processors that share a common memory. This approach would create a very effective organization for a limited number of processors. Thanks to CEs, this organization does not compromise extensibility in any way.

A possible organization consists of stations 1 and 2 sharing disk 1, while station 3 resorts to disk 2. If disk 1 did not exist for some reason, all stations could easily share disk 2. In any organization, any user would be allowed to log in at any station. Better disk performance obviously occurs if the user logs in as closely as possible to the disk(s) containing needed files. The upper CE connects to the outer world (where X terminals, for instance, can exist) and is shared by all stations.

Note that some stations can act as pools of unassigned processors to be used by whoever needs them. A station can migrate processes to another station, although with reduced priority if that station is assigned to a user. This nev-

ertheless increases throughput if that station's user is not logged in or is not using all assigned processors all the time (which is usually the case in interactive programming).

A conventional workstation corresponds to one station with one processor, one or more CEs (for Ethernet interface, graphics hardware, etc.) and one or more disks.

**Implementation status.** Machines such as the one briefly described do not exist yet. Very few proposals have been made for such a system, and most of them consist of conventional computers interconnected by a LAN or some high-speed network. The architecture most similar to DICE seems to be the one proposed by Feitelson,<sup>10</sup> which presents a multiuser parallel computer (not yet built). The basic difference is that Feitelson's computer has strict controls. A tree structure of controllers

closely oversees the PEs and coordinates the scheduling of processes (gang scheduling). DICE designers preferred to treat processes independently and to rely on a dynamic load-balancing algorithm to spread processes over available hardware. Control of Feitelson's architecture is hierarchical and related to the controllers' network, whereas control is truly distributed, without hierarchy, in DICE.

The SPAN project detailed specifications of a preliminary version of DICE and built a simulator that ran on transputers. A VLSI design of the DICE processor used standard cells for quick prototyping. Unfortunately, this design resulted in a 200-sq-mm die (with 2-mm technology); its fabrication was never pursued. The simulator was implemented to exactly mimic the hardware structure of the VLSI processor (so that the software could exercise the hardware bugs). Consequently, the simulation was unacceptably slow (about 7,000 instructions per second on an Inmos T800 transputer). Nevertheless, this speed was enough to build a kernel that provided basic services and ran a few demonstration programs written in Parle and compiled to the DICE machine code. (Further details appear elsewhere.<sup>11</sup>)

We chose to adopt an object-oriented model for the DICE processor because this would be a step toward providing hardware support for the increasingly popular object-oriented programming paradigm. In addition, the object organization, enforced by hardware through the use of tags, would provide a protection not possible with more conventional hardware.

The development of DICE continued after the SPAN project officially concluded. We are now designing a version to tackle



many points left open in the first simplified version. The features include

- distributed, shared, virtual memory with support for replicated (in several processors) objects with guaranteed consistency,
- increased protection, and
- support for common operating system features.

WE ARE CONSIDERING THE REIMPLEMENTATION of Mach (starting only from its interface specification), since a simple port would completely miss the capabilities of the hardware. (Mach is a parallel distributed operating system developed by Carnegie Mellon University.) A preliminary analysis indicates that this task is much easier than writing Mach for a conventional architecture, given the hardware support available. For example, Mach ports are protected by nature (the hardware provides capabilities).

We are now designing a new VLSI version of the DICE processor (this time using a full-custom VLSI circuit) and expect a preliminary chip to be working by the end of 1991. ■

## References

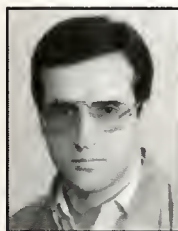
1. A.N. Refenes, P.C. Treleaven, and S. McCabe, "The ESPRIT SPAN Project: A Kernel System for Integrating Parallel Symbolic and Numeric Processing," Tech. Report TR-CS-149, Dept. Computer Science, University College London, 1988.
2. S. McCabe and M. Innes, "Parle: The Specification of the SPAN Target Machine Language," SPAN Report WP2-21, Thorn-EMI Central Research Laboratory, Hayes, London, 1988.
3. E. Eberbach, A.N. Refenes, and P.C. Treleaven, "The Structure of the VMC," Internal Report SPAN-WP1-15, Dept. Computer Science, UCL, 1988.
4. G. Roberts and R. Winder, "Integrating Symbolic and Numeric Computation," to be published in *Knowledge Engineering Review*, Cambridge University Press, Cambridge, England, June 1991.
5. G.A. Roberts, N. Wei, and R.L. Winder, "The Solve Object Oriented Programming System for Parallel Computers," SPAN-WP10-Deliverable-28, Dept. Computer Science, UCL, 1990.
6. P.A. Rounce, K. Chan, and M. Hardie, "Sprint: A Processor for a Parallel Architecture," *Proc. IFIP Int'l Conf. Very Large Scale Integration*, Elsevier Science Pub., Amsterdam, 1989, pp. 45-54.
7. P.A. Rounce et al., "VLSI Architecture Research Within the ESPRIT SPAN Project," *Microprocessing and Microprogramming*, Vol. 26, 1989, pp. 139-152.
8. B. Furht and V. Milutinovic, "A Survey of Microprocessor Architectures for Memory Management," *Computer*, Vol. 20, No. 3, Mar. 1987, pp. 48-67.
9. P.A. Rounce, "Final Report on the VLSI Workpackage of the SPAN Project," Tech. Report TR-CS-166, Dept. Computer Science, UCL, 1990.
10. D. Feitelson and L. Rudolph, "Distributed Hierarchical Control for Parallel Processing," *Computer*, Vol. 23, No. 5, May 1990, pp. 65-77.
11. J. Delgado et al., "Final Report on the DICE Architecture," ESPRIT Project SPAN, Report SPAN-WP6-41, Instituto de Engenharia de Sistemas e Computadores, Lisbon, Portugal, 1989.



**Peter Rounce** is a senior lecturer in Computer Science at University College London, where he teaches microprocessor systems architecture. His research activities concern parallel computer architectures, neural networks, and optical computing. His experience covers

software development from application programs to machine code, microcomputer hardware, digital electronic design using large-scale and medium-scale integration components, and VLSI design in silicon.

Rounce received a BSc degree in physics from Sussex University and a PhD degree in atmospheric physics from University College London.



**Jose Delgado** teaches computer science at the Instituto Superior Tecnico, Lisbon, Portugal. He also conducts research at the Instituto de Engenharia de Sistemas e Computadores in Lisbon. He started and now heads a group at the latter

institute working mainly on parallel architectures, object-oriented languages and systems, and artificial intelligence.

He is responsible for the ESPRIT Parallel Computing Action project 4121 (Parsec) and for the participation of Inesc in ESPRIT projects 1588 (SPAN) and 5404 (GPMIMD).

Delgado received the electronics engineer, MSc, and PhD degrees from the Instituto Superior Tecnico.

## Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 162

Medium 163

High 164





## Neurocomputing

*continued from p. 31*

```

/* Topology , Data & Function Information */
#define system_variable constant
struct (...) config = (...);
typedef struct (
    float state (constant); /* status info. */
    synapse_type *synapse; /* status info. */
    neuron_type **input_neuron; /* topological info. */
    neuron_type **outward_neuron; /* topological info. */
    rule_type weight_sum; /* functional info. */
    rule_type weight_update; ... /* functional info. */
)
    neuron_type;
struct (
    net_def; /* definition of network */
) system;

/* Control Information */
/* System function definitions */
connect ( ... )
read_weights (name_file) ( ... )
main ()
( /* calls to system functions to control application */
    connect ()
    read_weights (file_name)
    read_states (file_name)
    learn ()
    recall () ...
)

```

Figure 4. The NC framework for a neural network description.

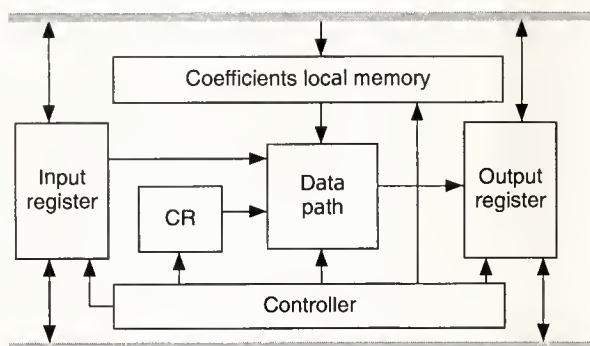


Figure 5. The neural processor. CR indicates a customizable register.

volving a cascadable neuron processor with local control and state value circulation through a shift register with no external control. We realized a 20-MHz chip with only one neuron; tests on it proved positive.

We can customize this architecture since its parameters are the number of neurons, layers, and interconnections; the state and coefficient precision; and the learning capabilities. Figure 5 depicts the neural processor. Such processors can be put together in a global architecture as shown in Figure 6.

## Image processing

We tried, in this project, to introduce neural techniques step by step in the image processing domain. We began to solve partial (high-level or low-level) problems. Next, we plan to develop coherent, complete applications for operation in our future ESPRIT project Galatea. Such applications would be capable of handling complex problems and showing the coupling of different neural networks and classical techniques to obtain a global result.

We thus consider the two years of Pygmalion as a first step of exploration in the domain of image processing.

We have chosen, among all the possible available application domains of image processing, two very different kinds of pictures. Let us expose their respective interests. The area of remote sensing data has several domains of applications: geology, agriculture, meteorology, hydrology, and cartography among others. Moreover, researchers have already studied this type of image with classical techniques, especially all areas that concern low-level processing; a lot of results are available for comparison. Nevertheless, processing this type of data is rather difficult, even by classical techniques. Moreover, we have no flexibility when dealing with these images. One is constrained by the precise view one is considering.

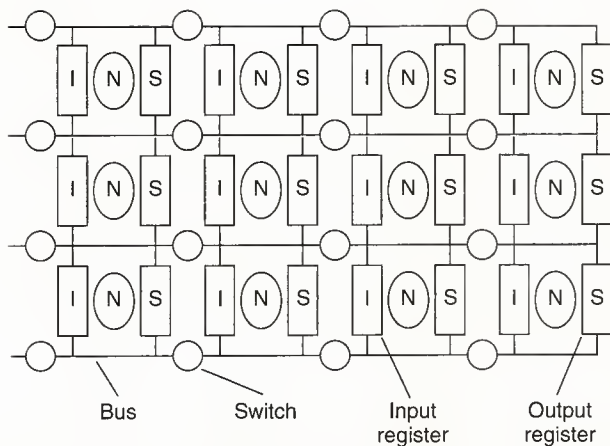


Figure 6. Putting together neural processors.

This is not the case with the other type of data we are considering, namely workpieces in a factory automation context. This problem is simpler and more tractable: One can easily choose the pictures one wants to deal with, according to the type of problem selected. We can roughly divide the different tasks to be investigated into low-level image processing performed on Spot images and high-level processing performed on workpiece images, after segmentation by classical techniques.

Low-level processing includes 1) compression using learning by back-propagation on a neural network in autoassociation and 2) segmentation into homogeneous regions via the combined approach of edge and region detection. Before the end of the project, we plan to perform a stereovision application.

We have only considered supervised segmentation. The use of the back-propagation algorithm to produce edges or to classify different textual regions, after extraction of local features, proved very efficient. High-level processing mainly concerns classification and pattern recognition in two and three dimensions.

For 2D, we considered two different approaches. In the first one we input a description of the object in terms of statistical or syntactical features to a neural network used as a classifier. During the learning process, the network evaluates the features and selects the most relevant ones. We compared several types of neural networks (Hopfield's network, the multilayer Perceptron, Kohonen's topological maps).

In the second approach we used the image directly (not preprocessed) to feed the neural network. The approach is based on a constrained multilayer Perceptron in which the first layer preprocesses the data. The synaptical coefficients are forced to be invariant with respect to translation and/or rotation of the objects presented to the network. Several

preprocessings can be done using different areas of the first layer. Then, the training algorithm (Back Propagation) defines the proper preprocessing(s) and performs the classification.

For 3D, we start with several images corresponding to a view of the same 3D objects from different angles. We want to be able to recognize automatically a 3D workpiece from a set of 3D objects, independently of their orientation or position on a plane. We use an associative memory algorithm to perform this task.<sup>1</sup> This application is still at a preliminary stage.

Finally, we compared these image-processing neural classifiers to more classical techniques, such as statistical methods and syntactical pattern-recognition techniques, for which we have available extensive experimental data.

### Speech and signal processing

Researchers have tried many heuristic and even sophisticated methods in learning about automatic speech recognition in the past. Whereas progress in many other fields of technology has been astonishingly rapid, research investments into this "natural" task have, however, not yet yielded adequate results. After initial optimism, researchers became more and more aware of the many difficulties to be overcome. Researchers place considerable hope in the application of artificial neural networks. They view it as a new method within this special area of pattern recognition that is capable of overcoming current problems and increasing recognition-system performance.

Noise interference, speaker insensitivity, and a large vocabulary are the main problem areas in current speech recognition. In traditional systems, features must be traded for other features to maintain a certain level of performance, for example, vocabulary size against speaker independence or against robustness in noise. In contrast, the characteristics of neural networks, as hitherto identified, led us to anticipate them. We'd like to provide such system features, thus offering an enlarged capability profile with at least comparable recognition rates. For an arbitrary selection of speech-recognition tasks, researchers have already experimentally demonstrated the efficiency of neural networks.

The objective of the speech processing application in the Pygmalion project is characterized by the investigation of a variety of artificial neural network architectures. These architectures—in accordance with appropriate training methods implemented by efficient learning algorithms for individual topics—provide several features. They are:

- 1) *Isolated word recognition.* This task focuses on speaker-adaptive isolated word recognition (IWR) for a medium-size vocabulary (about 100 words) in real office environments.
- 2) *Speaker independence and adaptivity.* We also plan to



study low-level preprocessing of speech signals with respect to feature extraction and noise reduction in relation to speaker-independent IWR for a limited vocabulary. We plan to show that IWR networks have feature extraction capabilities, which could be reused later on in a continuous speech recognizer.

- 3) *Speech signal preprocessing, including noise reduction.* Coding becomes a very important problem for speech recognition with neural networks. We must consider encoding the raw speech signals and extracting the relevant features, together with the form in which we plan to present the segmental (and possibly the suprasegmental) information.
- 4) *IWR in a noisy environment.* This task is devoted to speaker-independent recognition of isolated words in a telecommunications environment. It includes the design and implementation of a small vocabulary, isolated-word recognizer whose accuracy and robustness make it suitable for a telephonic application. Even if a very small vocabulary is used (for example, 10 digits plus some command words such as "help" and "repeat"), many possible automatic services could be offered to the telephone subscribers.
- 5) *Subword-unit recognition and coarticulation.* This task deals with discrimination, coarticulation, and subword units. A language parses speech into sensitive language units. If we consider that language as a compound of self-organized systems, it is very important to know how consonants and vowels self-organize and combine themselves into discrete units. This problem of coarticulation and subword units is a fundamental and critical aspect of multispeaker situations. The goal of this speech-processing research is to show that a nonarbitrary structure of discrete units exists inside words that permits a good multispeaker word speech recognition.

The last field of application we are exploring is the classification of underwater natural sounds. A relatively small study has tried to assess neural network capabilities compared to many kinds of algorithms already developed and optimized. We apply neurocomputing in two ways. We can use a processed version of the signal, obtained through well-known and efficient preprocessing algorithms. Or, we can directly apply neural classification to the raw signal.

This study approach also illustrates how efficient neural networks may be in terms of the amount of time and effort needed to develop an application. Provided we take into account the basic knowledge of the type of signal to be treated in the global structure of the network, automatic neural adaptivity is quicker to implement than theoretical studies. It is also more efficient than empirical research when significant characteristics must be extracted from the signal before

the recognition process. We expect this field to give rise to operational applications in the very near future.

SINCE THE JANUARY 1989 START of the Pygmalion project, we've made major progress on the environment and the applications.

We've produced the graphics monitor and algorithm library and fully specified the high-level N and intermediate-level NC languages.

We've made available for experimentation a first version of the graphics monitor (operating on an NC specification of a neural application) and the C version of the algorithm library. A compiler from N to C++ is also available. By the end of 1990, we project that the complete Pygmalion environment, including the final version of the graphics monitor, the compiler from N to NC, and a first version of the N library, will be finished.

We plan to reuse and complete this software environment in the ESPRIT II Galatea project; in particular, the development teams of Thomson, Philips, and Siemens plan to use it. Thus, we expect this tool to become a de facto European standard. A first commercial product should be available for sale by mid-1991.

The hardware integration study proved three points. Neurocomputing on silicon is feasible. Dedicated ASICs (application-specific ICs) can be generated in a short time starting from the building block that is the neuron processor. Customization of neural networks through silicon compilation techniques and PROM or soft-programmable switches should be studied in a future European project, namely the Galatea project.

We can state several general results from the applications study. For applications in which efficient classical preprocessing is known, results prove better with the use of preprocessed signals than with the raw signal. But when no efficient preprocessing is known, more effort must be exerted concerning the architecture of the network. Neural networks produce better results than classical methods every time the designers of the networks have taken advantage of their adaptive capacity. In other cases, the results roughly compare to the ones of the classical methods. In many cases, Back Propagation successfully and easily solved classification problems, even if some more sophisticated combinations of several algorithms have sometimes been allowed to outperform Back Propagation's results.

We've successfully implemented many image processing methods combined with classical methods, which will constitute the basic building blocks for the future operational applications to be developed in Galatea.

We demonstrated speech and signal processing methods, proving in particular noise independence for recog-

NEWLY RELEASED

## FORMAL VERIFICATION OF HARDWARE DESIGN

BY MICHAEL YOELI

This new book examines approaches for ensuring functional correctness of hardware through formal verification techniques prior to its production. It introduces the most important and promising techniques in formal hardware verification by concentrating on a number of relevant topics, rather than completely surveying the field. *Formal Verification of Hardware Design* deals with verification based on formal logic and discusses suitable, automated proof systems. Some of the key issues discussed throughout the tutorial include:

Basics of Formal Verification  
Formal Logic & Theorem Provers  
Hardware Description Languages

Abstraction Mechanisms  
Hybrid Approaches  
Timing Verification

340 pages. January 1991. Casebound. ISBN 0-8186-9017-8.  
Catalog No. 2017. \$62.00. Member \$50.00

MAIL YOUR ORDER TO:

**IEEE COMPUTER SOCIETY PRESS**  
**Customer Service Center**  
**10662 Los Vaqueros Circle**  
**Los Alamitos, CA 90720-1264**

OR CALL: 1-800-CS-BOOKS  
OR IN CALIFORNIA CALL: 714-821-8380

PLEASE ADD \$5.00 TO COVER HANDLING CHARGES

### Attention!

**CALL NOW FOR YOUR FREE  
IEEE  
COMPUTER SOCIETY  
PUBLICATIONS  
CATALOG**

Contains All The Latest  
IEEE CS Press  
Computer Science Titles in

- \* SOFTWARE
- \* NETWORKING
- \* ARCHITECTURE
- \* COMPUTER GRAPHICS
- \* STANDARDS
- & OUR NEW VIDEOTAPES

**Call: 1-800-CS-BOOKS**  
**or in California**  
**714-821-8380**

dition tasks. Even if fundamental difficulties are still an obstacle to efficient continuous speech recognition, industrial applications are already possible, in particular in the telecommunications environment. ■

### References

1. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.



**Bernard Angeniol** is one of the founders of Mimetics, a new company dedicated to neural network applications in industry. He previously headed the artificial intelligence department of the Electronic Systems Division of Thomson-CSF, which he set up. He also set up the ESPRIT project Pygmalion and the future Galatea project, which aims at building an European neurocomputer (three years, 17 million ECUs) beginning at the end of 1990. He cochaired the INNC-90 Paris conference on neural networks, which drew more than 1,000 participants.

Angeniol holds a PhD in mathematics and has authored more than 30 publications in mathematics and neural networks, including two books.

Direct questions concerning this article to the author at Thomson-CSF, Division Systemes Electroniques, BP 150, 92223 Bagneux Cedex, France.

### Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 165

Medium 166

High 167



## Micro View

*continued from p. 104*

sor (the 8085) and had done so; it had every reason to believe that it would have the opportunity to second-source Intel's successor product; but Intel failed to come through. In 1981 the tables were turned. Intel badly needed a second source for its 8086 microprocessor and had to come back to AMD. But AMD remembered the past. So the contract negotiated in 1982 whereby AMD would help Intel by second-sourcing the Intel 8086 was basically very favorable to AMD.

At the time, Intel needed an alternate source to bolster its competitive position against the Motorola 68000. AMD expected to focus its development efforts on peripheral chips, rather than processors. These designs would be transferred to Intel in exchange for Intel's processor designs. As the 8086 and 286 became successful, however, Intel no longer needed AMD. Furthermore, AMD's peripheral chip designs were late arriving and featured larger die sizes than anticipated. Consequently, the NIH (not invented here) factor caused Intel's product people to "look upon all AMD products with a jaundiced eye," according to Phelps.

Intel unilaterally decided to end their cooperation with AMD in 1984, but it kept this decision secret from AMD. Phelps noted that "Intel always thought the worst of AMD." He characterized Intel's actions as a classic example of a breach of contract—"preaching good faith, but practicing duplicity."

Phelps cited ineptitude rather than duplicity as AMD's main failing:

AMD management failed to appreciate the time required for and the enormous resources needed in both capital and personnel to change from LSI [large-scale integration] to VLSI

[very LSI] technology, and to adapt from NMOS [N-type metal-oxide semiconductor] to the CMOS [complementary MOS] process.

Ironically, Intel's developments contained their share of problems, but the company chose to focus its critical eye on AMD rather than itself.

### The "Big Case"

AMD attempted to argue that Intel's failure to uphold the agreement denied AMD the "benefit of the bargain." As a result, AMD contended it should receive the 386 design from Intel. Phelps dismissed this argument—called the Big Case—by stating that AMD assumed the contract between the two firms could force Intel to turn the product over to AMD in the absence of a value-for-value exchange. The judge termed this "an impossibility, both explicitly and conceptually."

Inexplicably, one page later, Phelps apparently contradicts his previous statement: "Whether AMD's losses caused by Intel's breaches of contract as found by this decision are sufficient to warrant a transfer of the 80386 to AMD will be determined in the remedies module."

The details of the particular product disagreements between Intel and AMD are too lengthy to describe here. A few aspects of Phelps's ruling, however, are worth highlighting.

The judge chided Intel's senior management for refusing to transfer the 8087 to AMD "even though a senior official at Intel in substantial charge of the AMD/Intel relationship believed Intel was legally obligated to transfer the part to AMD."

Intel also tried to persuade AMD to waive its rights to "CF points" (the way of measuring designs for the purpose of exchanges between the two companies) for a hard-disk controller product in return for the 8087. Phelps said, "AMD refused to give in to what it correctly perceived to be extortion on the part of Intel; and the 8087 has never

been transferred to this day."

Another interesting situation concerned the transfer of updates to the 286 design from Intel to AMD. According to the ruling Intel transferred "deliberately incomplete, deliberately indecipherable, and deliberately unusable" information. AMD resorted to reverse-engineering Intel's E-step and S-step versions of the 286 to continue to produce competitive products. Phelps called Intel's actions "inexcusable."

In one instance Phelps ruled that AMD is liable to Intel. In transferring the 7910 Codec chip to Intel, AMD subsequently developed a chip with new features—the 7911—instead of improving the 7910.

Intel contended that AMD breached the contract by failing to transfer the updates to Intel, but Phelps ruled that the improvements in the 7911 were not updates under the contract. Furthermore, he said that Intel cancelled its 7910 for internal reasons and not because of AMD's marketing efforts on behalf of the 7911. However Phelps ruled AMD *did* breach the contract by attempting to "bad mouth" the 7910 then being produced at Intel at the time AMD was in production with the 7911.

### Will Intel appeal?

Sadly, the waste of resources poured into this arbitration is not finished. The remedies phase promises to be as contentious, but presumably shorter, than the liabilities arbitration.

Even then, the litigation may not be over. Should Phelps award AMD the hundreds of millions of dollars the company is demanding in damages, Intel is likely to appeal.

### Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card

Low 186      Medium 187      High 188



## AMD v. Intel

**A**fter three years in arbitration, a judge recently rendered a decision regarding the ill-fated technology exchange pact between Intel and AMD. On October 11, retired superior court judge J. Barton Phelps's ruling only settled the issue of liability. A remedies module set for November 15 will address potential damages; this phase won't conclude until sometime next year.

Who won? The lawyers were the clear winners. At this point, they conducted 313 days of testimony, generated more than 42,000 pages of transcripts, and referenced 2,093 exhibits.

To an extent, AMD both won and lost. It won a moral victory—and perhaps a large cash settlement—when Phelps decided that Intel did not act in good faith and breached its contract with AMD. But since AMD previously said that it expected to receive rights to the 386 design as a result of the arbitration, AMD also lost.

In essence, Phelps ruled that Intel breached its contract with AMD, but it was not obligated to transfer any designs under the agreement because AMD failed to deliver acceptable designs in return. However, he left open the possibility that the remedies module may require Intel to transfer the 386 designs after all. This is inconsistent with the rest of the decision, which mandates a “product-for-a-product, value-for-value exchange.”

The judge's line of reasoning suggests that only a monetary settlement is possible. Intel counsel F. Thomas Dunlap made it clear in a conference call following the decision that Intel viewed any product transfer stemming from the remedies module as improper. According to Dunlap, Intel is likely to appeal if such a transfer is mandated.

The 386 transfer may become a moot point. At AMD's press conference following the decision,

chief executive officer Jerry Sanders stood in front of a plot of his firm's 386-compatible design—code-named “Longhorn”—and stated that AMD will announce the product regardless of the outcome of the arbitration.

The AMD 386 design uses Intel's microcode. AMD asserts that it holds a license to use any Intel microcode as part of a 1976 agreement; Intel maintains that the agreement gives AMD the right to copy, but not to distribute, the microcode. Pending litigation regarding the 287 math coprocessor will settle this issue.

The outcome of the microcode litigation could be far more important than the current arbitration. AMD presumably developed a clean-room version of the microcode, but it is likely to contain compatibility problems. If AMD loses the microcode copyright case, and if it doesn't receive rights to the 386 in the remedies module, the company will have to pull the part off the market and face a suit by Intel for any profits made from selling the chip.

### Management at fault

The arbitration decision was a stinging indictment of the management of both companies. Judge Phelps concluded the situation arose from many causes. As competitors, Intel and AMD found it difficult to adjust to a contract calling for mutual cooperation. Other contributing factors included a contract that heavily favored AMD and the failures of senior management on both sides to carry out their contract responsibilities.

The judge offered some background leading to the ill-fated 1982 agreement between the firms:

In the late 1970s, AMD was left hanging out to dry by Intel. AMD had agreed to second-source Intel's 8-bit microproces-

*continued on p. 103*

Michael Slater

Microprocessor Report

(707) 823-4004

mslater@cup.portal.com



## USE THE READER SERVICE CARD TO OBTAIN INFORMATION ON:

- Membership application, student #203, others #202
- Periodicals subscription form for individuals #200
- Periodicals subscription form for organizations #199
- Publications catalog #201
- Comppmail II electronic mail brochure #194
- Technical committee list/application #197
- Chapters lists, start-up procedures #193
- Student scholarship information #192
- Volunteer leaders/staff directory #196
- IEEE senior member grade application #204

(requires ten years practice and significant performance in five of those ten)

To check membership status or report a change of address, call the IEEE toll-free number, 1-800-678-4333. Direct all other Computer Society related questions to the Publications Office.

## PURPOSE

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

## MEMBERSHIP

Members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.



IEEE COMPUTER SOCIETY

A member society of the  
Institute of Electrical and Electronics Engineers, Inc.

## PUBLICATIONS AND ACTIVITIES

**Computer.** An authoritative, easy-to-read magazine containing tutorial and in-depth articles on topics across the computer field, plus news, conferences, calendar, interviews, and product reviews.

**Periodicals.** The society publishes six magazines and five research transactions. Refer to membership application or request information as noted above.

**Conference Proceedings, Tutorial Texts, Standard Documents.** The Computer Society Press publishes more than 100 titles every year.

**Standards Working Groups.** Over 100 of these groups produce IEEE standards used throughout the industrial world.

**Technical Committees.** More than 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

**Conferences/Education.** The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

**Chapters.** Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.

## OMBUDSMAN

Members experiencing problems — magazine delivery, membership status, or unresolved complaints — may write to the ombudsman at the Publications Office.

## EXECUTIVE COMMITTEE

President: Helen M. Wood\*  
National Oceanic and Atmospheric Administration  
FB 4, Rm. 1069, Code E/SP  
Washington, DC 20233  
(301) 763-1564

President-Elect: Duncan H. Lawrie\*  
Past President: Kenneth R. Anderson\*

VP, Conferences and Tutorials: Laurel V. Kaleda (1st VP)\*  
VP, Standards: Paul L. Borrelli (2nd VP)\*  
VP, Area Activities: Gerald L. Engel†  
VP, Education: Ronald G. Hoelzeman†  
VP, Membership and Information: Barry W. Johnson†  
VP, Press Activities: James H. Aylor†  
VP, Publications: Sallie V. Sheppard\*  
VP, Technical Activities: Mario R. Barbacci\*

Secretary: David Pessel\*  
Treasurer: Joseph Boykin\*  
Division V Director: Edward A. Parrish, Jr.†  
Division VIII Director: J. T. Cain†  
Executive Director: T. Michael Elliott†  
\*voting member of the Board of Governors  
†nonvoting member of the Board of Governors

## BOARD OF GOVERNORS

### Term Expiring 1990:

Vishwanth Agrawal, Mario R. Barbacci,  
Ming T. (Mike) Liu, Yale N. Patt, Donald E. Thomas,  
Benjamin W. Wah, Ronald Waxman

### Term Expiring 1991:

P. Bruce Berra, Michael Evangelist,  
Ted Lewis, Raymond E. Miller, Earl E. Swartzlander, Jr.,  
Joseph E. Urban, Thomas W. Williams

### Term Expiring 1992:

Alicja I. Ellis, Tadao Ichikawa,  
David Pessel, Sallie V. Sheppard, Bruce D. Shriver,  
Harold Stone, Wing N. Toy

### Next Board Meeting

March 1, 1991, 8:30 a.m.  
Cathedral Hill Hotel, San Francisco, CA

## SENIOR STAFF

Executive Director: T. Michael Elliott  
Publisher: H. True Seaborn  
Director, Conferences and Tutorials: Anne Marie Kelly  
Director, Finance and Administration: Tod S. Heisler  
Director, Board and Administrative Services: Violet S. Doan

## COMPUTER SOCIETY OFFICES

### Headquarters Office

1730 Massachusetts Ave. NW  
Washington, DC 20036-1903  
Phone: (202) 371-0101  
Fax: (202) 728-9614

### Publications Office

10662 Los Vaqueros Cir.  
PO Box 3014  
Los Alamitos, CA 90720-1264  
Membership and General Information:  
(714) 821-8380  
Publication Orders: (800) 272-6657  
Fax: (714) 821-4010

### European Office

13, Ave. de L'Aquilon  
B-1200 Brussels, Belgium  
Phone: 32 (2) 770-21-98  
Fax: 32 (2) 770-85-05

### Asian Office

Doshima Building  
2-19-1 Minami-Aoyama, Minato-ku  
Tokyo 107, Japan  
Phone: B1 (3) 408-3118  
Fax: B1 (3) 408-3553



## IEEE OFFICERS

President: Carleton A. Bayless  
President-Elect: Eric E. Sumner  
Executive Vice President: Martha Sloan  
Secretary: Fumio Harashima  
Treasurer: Wallace S. Read

VP, Educational Activities: Richard S. Nichols  
VP, Professional Activities: Michael J. Whitelaw  
VP, Publication Activities: Ralph W. Wyndrum, Jr.  
VP, Regional Activities: Robert T. H. Alden  
VP, Technical Activities: H. Troy Nagle, Jr.

# Discover Parallel Processing

## Quadputer™

The Microway Quadputer is the world's most popular PC Transputer development environment. It can be purchased with two to four Transputers and one to four megabytes of RAM per processor. The Quadputer runs all the popular Transputer development software, all of which is available from Microway. It is compatible with our Monoputer™ which provides 1 to 16 megabytes of RAM and a single T800, our Videoputer™ which comes in VGA and higher resolution versions and is powered by a memory mapped pair (T800 and 34010), and our Linkputer™ whose cross bar switching network can dynamically link up to 32 Transputers. Finally, all Microway Transputer products can be used with our Number Smasher-860 to provide out-of-this-world numeric performance!

For more information, please call 508-746-7341.

# Microway

The World Leader in PC Numerics

Corporate Headquarters, Research Park, Box 79, Kingston, MA 02364  
TEL 508-746-7341 • FAX 508-746-4678  
U.K. - 32 High St., Kingston-Upon-Thames, 081-541-5466 • Italy 02-74.90.749  
Holland 40 836455 • Germany 069-75-2023 • Japan 81 3 222 0544

## Number Smasher® 860

The highest performance coprocessor card to ever run in a PC, Number Smasher-860 delivers up to 80 million single precision floating point operations per second at 40 MHz and produces over 10 Linkpack megaflops. The board comes standard with an ISA interface, two Transputer Link Adaptors that allow it to interface with a Microway Quadputer or Videoputer, your choice of our NDP Fortran, C or Pascal for the 80860, plus 8 megabytes of high speed memory.

## NDP Fortran-860, C-860 and C++-860

Microway NDP 860 Compilers make it easy to recompile your favorite mainframe, 80386 or PC application for the 80860. The resulting code runs on our XTEND-860™ environment under DOS, UNIX or XENIX.

